Universitat Autònoma de Barcelona

Degree Final Project

Creation of Graphical User Interfaces for R

Sensitivity study of a model

Author:
Miriam Amores Gago

Tutors:
Mercè Farré
Aureli Alabert

Year 2012-2013

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Since the beginning of 2012 there is a project between mathematicians and veterinarians of Universitat Autònoma.

This project involves the study of the sensitivity analysis of a response variable (profit) with respect to various input parameters and protocols used to describe the life of the cows and the milk production in a dairy farm. In the long term, this project will contribute to the efficient management of the farms. To this end, the time at which a cow has to be replaced by another one is the key parameter to analyze.

The analytical treatment was discarded due to the complexity of the model. Alternatively, we chose the replicable simulation of a model with random and deterministic variables. The task of optimizing the time when a cow is to be replaced is complex because it involves many factors. The sale of a cow implies some benefit, buying a new cow has a certain cost and the net income that produces a cow over its life is difficult to predict. Since many factors (diseases, accidents, etc.) affect the production along the cows life, the expected production of a new cow (by its race, condition, origin and others) must be modified along the time after observing its real production in previous time intervals. A first step is simulating the process and repeating it many times in order to make statistical analysis on this data looking at different parameters that may have an influence.

Once the simulation and replication process are designed and the code has already been created in an appropriate language we must consider how to display the results. It is important that a typical user (scientist, entrepreneur, etc.) can use it easily and analyze the results, according to his/her needs.

The code created by mathematicians in the initial project was programmed with R, a freely available language and environment for statistical computing and graphics.

Since the objective is to see how different parameters affect the model output, users must be able to quickly change parameters and check out the output easily, without changing (or even seeing) the R code.

This problem can be solved by creating graphical user interfaces (GUIs) adapted to the model checking. With a GUI, everybody would be able to interact with the program by introducing a few number of initial parameters, without any knowledge of the programming language.

We have designed two kinds of GUIs. The first kind is simpler than the second one because it involves few parameters and we have used it as a learning step for building the final GUI. In this first step, we built GUIs that serve us to check the model basis. Specifically, they apply to analyze four variables of special interest.

These variables are random times that must fit, in an approximate way, the reproductive cycles of a cow. We have created four GUIs to estimate well these random variables because they play a crucial role in the global model. As mathematicians, we aren't able to compare real and simulated cows behaviors, this task correspond to the veterinarians.

The second and final step of this project was to create the final GUI, which encompasses all parameters and variables needed to estimate the farmer's profit.

1.1 A BRIEF INTRODUCTION TO THE COW'S INDUSTRY

A cow gives profit if it produces a considerable quantity of milk. If the cow does not give a sufficient quantity of milk, its sustenance will suppose to the farmer a cost bigger than the earnings obtained for selling its milk. In this case, the farmer should decide to sacrifice the cow in order to have earnings selling its meat instead of losing money if no action is done.

Each cow is inseminated and is expected to get pregnant. If it does not get pregnant, the farmers will wait for the next ovulation (around 21 days) to carry out the insemination again. Once the calf is born, the cow produces milk during a few months, and this is what gives benefit to the farmer. Some days later, the process of insemination starts again to begin a new reproductive cycle of the cow.

Each farm follows a protocol to decide when it is time to replace or not a cow. A protocol is defined by certain thresholds, dates and alarms which indicate when a cow stops being profitable. When a cow is not pregnant after a few inseminations, it is dried, sold, a benefit is obtained for its meat and a new cow is bought. Deciding the optimum point at which a cow generates more expense than profit is one of the points that is wanted to be determined at long term.

We will focus on a specific protocol, called *Trebol*, which provides daily checks of the milk production and the state of cows, starting at date t_0 (100 days) from the beginning of the cycle. If the cow is not pregnant and its production does not exceed the threshold L_1 (15 liters, for example), the sacrifice is immediate. In fact, if the cow is below a higher threshold L_2 (about 25 liters) and their production genetic potential is low, insemination stops and the veterinarian programs the sacrifice of the cow when the threshold L_1 will be reached.

Deciding the optimum point at which a cow generates more expense than profit is one of the points that is wanted to be determined at long term.

What it is wanted to be studied first is the dependencies on the first threshold L_1. This will be the main objective of this project, provide the necessary tools to see the evolution of the daily benefit of a farm according to the thresholds. At long term this will help to achieve an efficient management of a dairy farm.

In the following sections we explain what is exactly a graphical user interface, why it is needed, which are their pieces and how to construct one.

Chapter 2

THE GRAPHICAL USER INTERFACE (GUI)

2.1 INTRODUCTION

A GUI is the primary means of interaction with desktop environments. User controls are packed into hierarchical drop-down menus, buttons, sliders, etc. and the user manipulates the windows, icons, and menus with a pointer device, such as a mouse.

Usually, a GUI is based on menus and dialog boxes,but there are different ways to organize the commands and to display the results. A basic example of a common GUI that is used by everybody is the dialog box that appears in the screen when one wants to print something. When the "print" button is pressed it appears a GUI collecting arguments from the user to customize the printing of a document. Below there are three examples of GUIs that could appear in the case described.

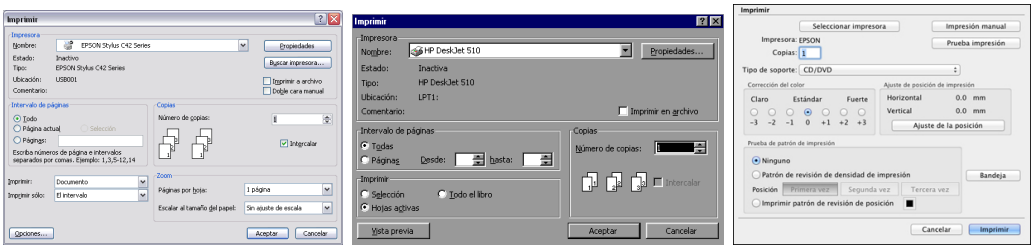

Figure 2.1: Different GUIs showing different options to customize the printing of a document.

In the pictures one can see that the three GUIs perform the same task but they have different appearance. This is because there is a wide variety of available widget types, and widgets may be combined in an infinite number of ways. Then, there are often numerous means to achieve the same goals, as in the case of configuring a printout. It also depends on the library of code with

which one works and the operating system in which one is that a GUI has one appearance or another. In the example above the first and second GUIs are from different versions of Windows operating system, and the third one is from Mac operating system.

One of the main features that has to fulfill a GUI is to be clear and intuitive, so that anyone will be able to use it.

The creation of GUIs is the perfect solution to play with a function that depends on different parameters, because a GUI allows anybody to see immediate results of a function changing her different parameters by simply clicking some widgets. And one of the main advantages is also that one does not need to know the language with which the program is made to use the interface and interpret their results.

Because of all these advantages a GUI is the perfect solution for a common user to interact with parameters and look at the variations on the results.

2.2 TYPES OF GUIs

Two common types of user interfaces in statistical computing are the command line interface (CLI) and the graphical user interface (GUI). The usual CLI consists of a textual console in which the user types a sequence of commands at a prompt, and the output of the commands is printed to the console as text.

As an example, R is provided with a command line interface (CLI), which is the preferred user interface for power users because it allows direct control on calculations and it is flexible. However, good knowledge of the language is required.

Also, R, as a GNU project (General Public License), offers the opportunity to develop and test various different GUIs. There are several R packages to create GUIs. Different packages has to be proved, learning how to make easy GUIs with each one, in order to decide which will be better to develop a concrete work. It is also important for a GUI to be efficient and easy to program.

R users that are familiar with R programming and knows the code rarely will need to create a GUI because surely they would prefer to introduce the code directly in R and see the results. But for those who doesn't know how to program with R and have no intention of learning, a task-specific GUI will be very useful. These interfaces do not usually contain a command line, as the limited scope of the task does not require it.

To create a GUI we need some package or library with good functions providing different widgets and a way to combine these widgets to construct a GUI. It must have a search about the existent packages created to program graphical user interfaces with R in order to choose the proper one to make the appropiate GUI.

2.3 THE STRUCTURE OF A GUI

We are going to explain which is the structure of a GUI. Almost every GUI has a hierarchical layout.

The primary component is the window. A typical interface design consists of a top-level window referred to as the *document window* that shows the current state of a "document", that in R could be a data frame, a command line, a function editor, a graphic or another complex form. The user executes commands, often called actions, on a document by interacting with graphical controls. Every control in a window belongs to some abstract menu. Two common ways of organizing controls into menus are the menu bar and toolbar. The top-level window forms the root of a hierarchy, and is usually called a *container*.

The parameters of an action call are controlled in sub-windows. These sub-windows are termed *dialogs*, or *dialog boxes*. These terms may also refer to smaller sub-windows that are used for alerts or confirmation. The program often needs to wait for user input before continuing with an action, in which case the window is modal. We refer to these as *modal dialog boxes*.

Each window or dialog typically consists of some controls laid out in some manner to facilitate the user interaction. Each window and control is a type of widget, the basic element of a GUI, every GUI is constituted by its widgets.

2.4 TYPES OF WIDGETS

2.4.1 CONTROLS

A GUI comprises one or more widgets. The appropiate choice depends on a balance of considerations. For example, many widgets offer the user a selection from one or more possible choices. An appropiate choice depends on the type and size of the information being displayed, the constraints on the user input, and the space available in the layout.

There is a wide variety of available widget types, and, as we have seen in the example below, widgets may be combined in an infinite number of ways to achieve the same goals. Depending on what we want to do we will use one

type of widgets or another one.

In table 2.1 we can see different types of widgets used for differents purposes depending on the type and size of data and the number of itemps to select.

Type of data	Single selection	Multiple selection
Boolean	checkbox, toggle button	-
Small list	radio button group, combo box, list box	checkbox group, list box
Moderate list	combo box, list box	list box
Large list	list box, auto complete	list box
Sequential	slider, spin button	-
Tabular	table	table
Hierarchical	tree	tree

Table 2.1: Possible selection widgets by data type and selection mode.

In the next figure 2.2 we see a simple GUI that shows several controls in a single dialog. This GUI shows a dialog for specifying different components in a model. A checkbox enables an intercept, a radio group selects either full factorial or a custom model, a combo box selects the "sum of squares" type, and a list box allows for multiple selection from the available variables in the data set.

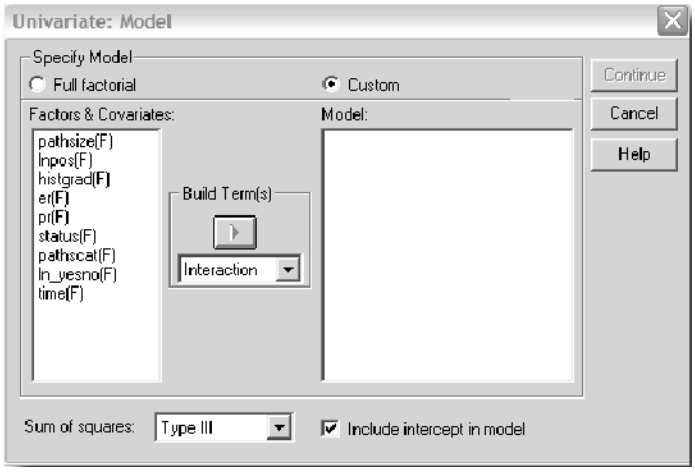

Figure 2.2: A dialog box for specifying components of a linear model.

The figure 2.2 shows a dialog that allows the user to specify individual terms in the model using several types of widgets for selection of values, such as a radio button group, a checkbox, combo boxes, and list boxes.

2.4.2 PRESENTING OPTIONS

The widgets that receive user input need to translate that input into a command that modifies the state of the application. Commands, like R functions, often have parameters, or options. There is a list of possible widgets that we can choose depending of what we want to create, let's introduce them briefly:

- **Checkboxes:** A *checkbox* specifies a value for a logical (Boolean) option. Checkboxes have labels to indicate which variable is being selected. Combining multiple checkboxes into a group allows the selection of one or more values at a time, or none.

- **Radio buttons:** A *radio button group* selects exactly one value from a vector of possible values. When a new button is pushed in, the previously pressed button pops out. Radio button groups are useful, provided there are not too many values to choose from, as all the values are shown. These values can be arranged in a row, a column or both rows and columns to better fill the available space.

- **Combo boxes:** A *combo box* is similar to a radio button group because it is used to select exactly one value from several ones. However, a combo box displays only the value currently selected, which reduces visual complexity and saves space, at the cost of an extra click to show the choices. A combo box is generally desirable over radio buttons when there are more than four or five choices.

- **List boxes:** A *list box* displays a list of possible choices in a column. While the radio button group and combo box select only a single value, a list box supports multiple selection. Another difference is that the number of displayed choices depends dynamically on the available space. If a list box contains too many items to display simultaneously, a scroll bar is typically provided for adjusting the visible range.

- **Sliders:** A *slider* is a widget that selects a value from a sequence of possible values, typically through the manipulation of a knob that moves or "slides" along a line that represents the range of possible values. The slider is a good choice for offering the user a selection of ordinal or numerical parameter values.

- **Spin buttons:** A *spin button* plays a role similar to the slider because it selects a value within a set of bounds. Typically, this widget is drawn with a text box displaying the current value and two arrows to increment or decrement the selection. Usually, the text box can also be edited directly. A spin button has the advantage of using less screen space, and offers the possibility of entering a specific value, if known, which is easier than selecting it with a slider. One disadvantage is that the position of the selected value within the range is not as obvious as with the slider.

The same message could be implemented through different widgets. For example, in figures 2.3 we can see how the same message "`Hello World`" is shown by different widgets.

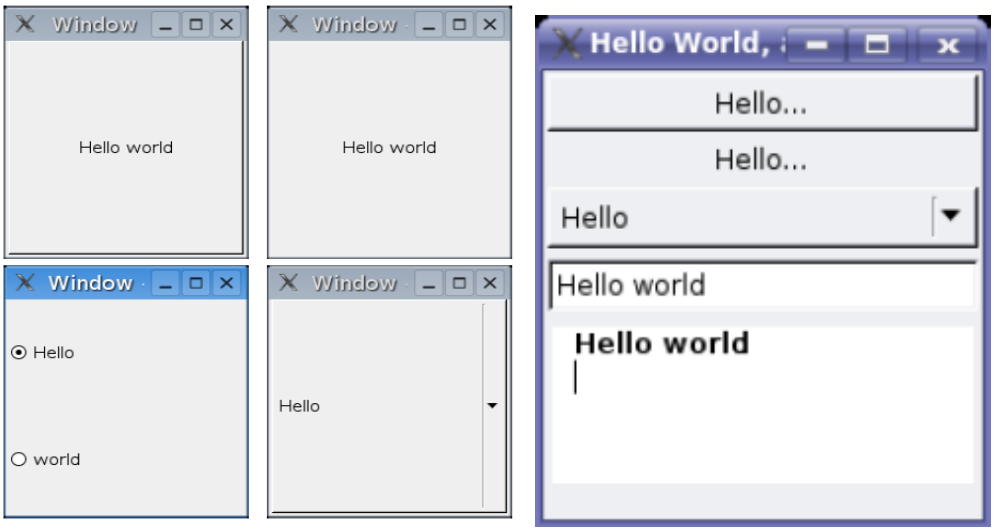

Figure 2.3: In the first image there is "Hello World" on a button, a label, a radiobutton and a list box. In the second one it is too in a blank and an editable space.

2.4.3 INITIATING AN ACTION

After a user specifies the parameters of an action, typically by interacting with the selection widgets presented above, it is time to execute the action through the next widgets:

- **Buttons:** A *button* issues commands when invoked, usually via a mouse click.
- **Icons:** An *icon* is used to complement or replace a text label on a button or other control. A button represents an action, so an icon on a button should visually depict an action.

  As an example we can see in the figure 2.4 the same action of `Refresh`in both widgets, a button and an icon.

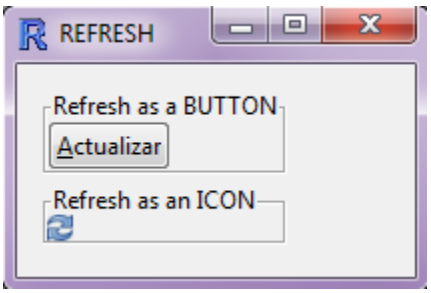

Figure 2.4: The action of Refresh on a button and on an icon.

- **Menu bars:** The *menu bar* contains items for many of the actions supported by the application. By convention, menu bars are associated with a top-level window. To help the user navigate, menu items are usually grouped with either horizontal separators or hierarchical submenus.

- **Toolbars:** The *toolbars* are used to give immediate access to the frequently used actions defined in the menu bar. Toolbars typically have icons representing the action and perhaps accompanied by text.

- **Action objects:** An *action object* is essentially a data model, with a widget acting as a view. Common components of an action include a textual label, an icon, perhaps a shortcut, and a handler to call when the action is selected.

2.4.4 OTHERS

There are other widgets that are also important but will not be used in this project.

First, there are *modal dialogs*, that are dialogs that keep the focus until the user takes an action to dismiss the box. There are different types, as *modal dialog boxes, message dialogs* or *file choosers*.

Secondly there are several forms of displaying data, as in a *table widget*, that shows tabular data, or *tree widgets*, that shows data with a hierarchical structure.

Also to display and editing text there are *single lines of text* and *text-editing boxes*.

And finally there are widgets that display information but do not respond to user input. Their main purpose is to guide the user through the GUI and to display feedback and status messages, for example *status bars, progress bars* and *tooltips*.

2.5 SELECTING A CONVENIENT GRAPHICS LIBRARY

Four possible packages used for creating GUIs with R were studied, and we had to choose which one would be better to work with. The four candidates were:

- **Qtbase:** *Qt* is an open-source C++ library from Nokia. The R package *qtbase* provides an interface from R to Qt. As Qt is implemented in

C++, it is designed around the ability to create classes that extend the Qt classes. Qtbase supports this from within R, although such object-oriented concepts may be unfamiliar to many R users.

- **R-Tcl/Tk:** the *tcltk package* interfaces with the Tk libraries. Although not as modern as GTK+ or Qt, these libraries come preinstalled with the Windows binary, thus bypassing any installation issues for the average end-user. The bindings (see appendix A) to Tk were the first ones to appear for R and a lot of GUI projects use this toolkit (see appendix A).

- **RGtk2:** the *RGtk2 package* provides a link between R and the GTK+ library. GTK+ is mature, feature rich, and widely used.

- **RwxWidgets:** The *RwxWidgets* package provides an R-language interface to the wxWidgets GUI toolkit. The wxWidgets toolkit is a cross-platform, native toolkit in that it can be used on all major platforms and uses the native look-and-feel of each of those platforms.

The first one was an application framework known for its collection of GUI widgets and developed by Nokia, but the idea of programming with this package was rejected because there was very little information about it and there was not enough examples.

The second one was a Tool Command Language (TCL) which allows the use of a Tool Kit (see A) which add to the instructions of Tcl instructions to create a GUI. Programming with this package was rejected because, despite it was easy to program with it, it is said that it is not very efficient and that is slow to execute (see references [1]). As a good point the the code that this library uses is brief, but, as it happens with *Qtbase*, there was not enough information of this package to be able to create a GUI from zero.

The two remaining packages were RGtk2 and RwxWidgets. It seemed that both had good tutorials and that it will be possible to program with this packages, with some more reading and some work with examples.

We decided trying to start programming a simple GUI with these two packages. The idea was to program a function in R with different parameters, as those generating Gaussian values for example, and make a first simple GUI with it in order to get familiar with the functions of each package.

Doing the first basic GUIs with RGtk2, we realized that this package requires some amount code to do basic instructions. To create a simple dialog-box several lines of code were required and the code follows a rather complex structure. Then, trying to find information about RwxWidgets we found information about two new packages to create R GUIs and it was decided to also

try them.

The first one was very interesting because it was a package that quickly creates a GUI for a given function by automatically creating widgets for arguments of the function. The interface was essentially a wrapper (see appendix A) to some tcltk routines to simplify GUI creation. This package was *fgui*. It was useful because of its speed to create a simple GUI for a certain function but it was useless creating graphics or deciding where to put what. For example, if one wants to create a function that depends on some parameters, the package *fgui* creates a GUI where the values of the parameters of the function can be input but it is not able to plot the function, it just show the value of the function evaluated at one point. Clearly this wasn't what was needed for the project. This problem, together with the fact that it could not be decided where to put the widgets or the menu were crucial to discarted working with this package.

In tables 2.2 and 2.3 and corresponding figures 2.5 and 2.6 we can see two examples created with the package *fgui*:

```
library(fgui)
fguiWindowPrint("Goes to the console because no window has been
     created.")
mgui(rgeom,title=c("Random","Geometric"))
mgui(rbinom,title=c("Random","Binomial"))
fguiNewMenu(c("Random","SEPARATOR")) # Puts a separator in the menu
mgui(rnorm,title=c("Random","Normal"))
mgui(runif,title=c("Random","Uniform"))
fguiWindowPrint("Goes to the main window, now that it has been
     created.")
```

Table 2.2: Code to create a menubar with the options of creating randomly numbers of a geometric, binomial, normal or uniform distribution with *fgui*.

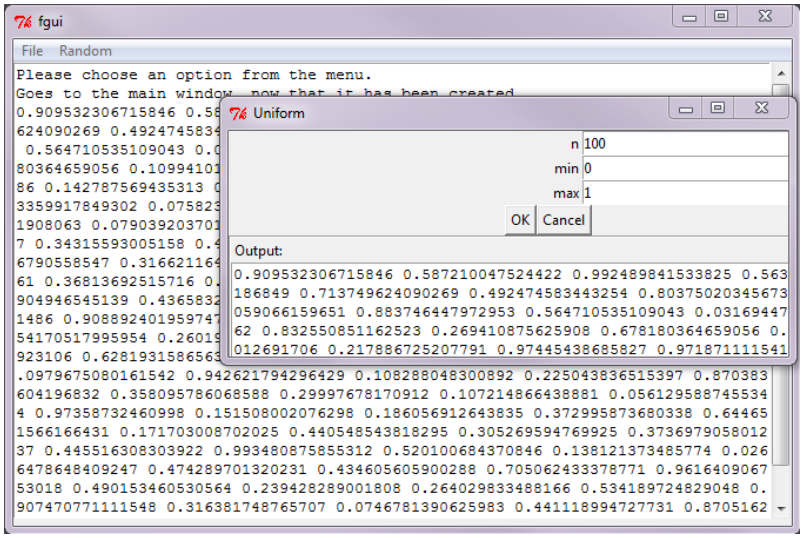

Figure 2.5: Example of a GUI that creates 100 numbers following a uniform distribution.

```
library(fgui)
f<-function(A,B,C,t=(0:350))A*t^B*exp(-C*t)
gui(f)
```

Table 2.3: Code to create a GUI of the Wood function (see chapter 3.1) with *fgui*.

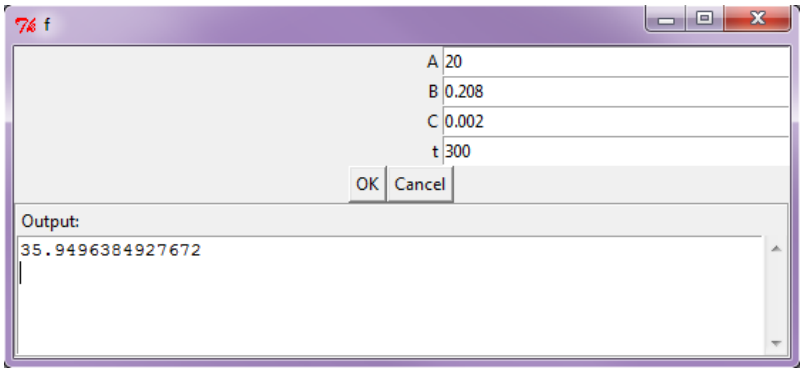

Figure 2.6: GUI of the Wood function that depends on A, B, C and the time.

The other package that we found was *gWidgets*. The gWidgets interface is much simpler than the original toolkits, and it uses a very intuitive hierarchical order.

In figure 2.7 we have a good example in which we can see the plot of a density function in which the user can choose the values of several options. In table 2.4 we can see the code to create this GUI.

```
library(gWidgets)
# types of distributions
availDists<-c(Normal="rnorm", Exponential="rexp")
# types of kernels
availKernels<-c("gaussian", "epanechnikov", "rectangular",
"triangular", "biweight", "cosine", "optcosine")
# creation of the function that will be executed
updatePlot<-function(h,...)  {
     x<-do.call(availDists[svalue(distribution)],
     list(svalue(sampleSize)))
     plot(density(x, adjust = svalue(bandwidthAdjust),
     kernel = svalue(kernel)),main="Density plot") rug(x)}
# creation of radio buttons for the distribution
distribution<-gradio(names(availDists), horizontal=FALSE,
     handler=updatePlot)
# creation of a combo box for the kernel
kernel<-gcombobox(availKernels, handler=updatePlot)
# creation of a slider for adjustl
bandwidthAdjust<-gslider(from=0,to=2,by=.01, value=1,
      handler=updatePlot)
# creation of a radio buttons for the sample size
sampleSize<-gradio(c(50,100,200, 300), handler = updatePlot)
# creation of the main window
window<-gwindow("gWidgetsDensity")
# creation of the main container
BigGroup<-ggroup(cont=window)
# creation of a subcontainer
group<-ggroup(horizontal=FALSE, container=BigGroup)
# adding all the widgets to the container
tmp<-gframe("Distribution", container=group) add(tmp, distribution)
tmp<-gframe("Sample size", container=group) add(tmp,sampleSize)
tmp<-gframe("Kernel", container=group) add(tmp,kernel)
tmp<-gframe("Bandwidth adjust", container=group) add(tmp,
     bandwidthAdjust,expand=TRUE)
# adding a space for graphics to the main container
add(BigGroup, ggraphics())
```

Table 2.4: Code to create a GUI that shows you the plot of the distribution that you choose selecting various parameters with *gWidgets*.

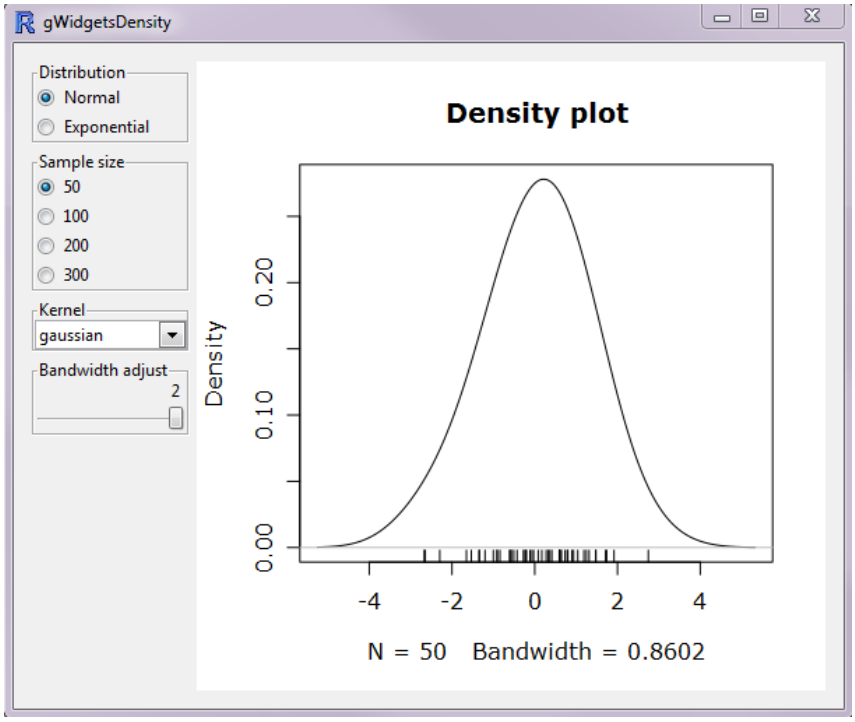

Figure 2.7: GUI of a Normal or Exponential distribution created with *fgui*.

As we can see, this package is not difficult to use. One can choose the type of widget for every parameter and decide where in the principal window they will appear. In gWidgets tutorials it is shown that a lot of options are possible (see references [4]). It is for these reasons that the GUI for this project has been built with the package *gWidgets*.

2.6 GWIDGETS PACKAGE

`gWidgets` is intended to be a cross platform means within an R session to interact with a graphics toolkit. It can be used through `RGtk2`, `tcltk`, `Qt`and `rJava`.

The `gWidgets` package provides a convenient toolkit to create rapidly medium-size GUIs within R. The package provides different instrucions to allow users to create several widgets. Unlike the other toolkits, gWidgets has relatively few constructors and methods.

2.6.1 CONSTRUCTORS

The widgets are all produced by calling the appropriate constructor. In the gWidgets library most of these constructors have the following basic form:

```
gname(some_arguments, handler = NULL, action = NULL,
      container = NULL, ..., toolkit = guiToolkit())
```

where `some_arguments` varies depending on the object being made.

This is a list of the common arguments used:

- The **handler** and **action** arguments: these are the instructions that are executed through event handlers (see section 2.6.3).
- The **container** argument: A top-level window does not have a parent container, but the other GUI components do. In gWidgets the parent container is passed to the widget constructor through the container argument. This argument name can always be abbreviated **cont**.
- The `...` argument: is the mechanism used to pass other additional arguments specific to each widget.
- The **toolkit** argument: it is usually not specified. It allows the user to mix packages within the same R session, but it can cause problems due to incompatibilities between packages. The default instruction for the toolkit argument is to call `guiToolkit`. This function will check wether a toolkit has been specified, if it is not, then a menu will be provided to the user to choose one. In this project the toolkit used is declared at the beginning of the code, then it is no necessary to specify the toolkit in every argument.

  Here we can see an example of how this function is used.

```
options(guiToolkit = "RGtk2")
```

The constructors produce one of three general types of widgets:

- **Containers:** such as a top-level **window**, a **paned group** or a **frame**.
- **Components:** such as an unnamed **label**, an **edit area** or a **button**.
- **Dialogs:** such as galert and gfilebrowse.

2.6.2 CONTAINER WIDGETS

After identifying the underlying data to manipulate and deciding how to represent it, GUI construction involves three basic steps:

- creation and configuration of the main components,
- the layout of these components, and

- connecting the components through callbacks to make a GUI interactive.

Let us remember that a GUI is built by placing components (child objects) in parent containers, and these, in turn, in other parent containers, and so on.

The `gWidgets` package provides just a few types of containers: toplevel windows (`gwindow`), box containers (`ggroup, gframe, gexpandgroup`), a grid container (`glayout`), a paned container (`gpanedgroup`), and a notebook container (`gnotebook`).

In appendix B we can see in a clear way all the diferent widgets for gwidgets package. In tables 2.6 and 2.5 we can see the different constructors that could be used to built a GUI:

Constructor	Description
glabel	A text label.
gbutton	A button to initiate an action.
gcheckboxgroup	A group of checkboxes.
gradio	A radio button group.
gcombobox	A drop-down list of values, possibly editable.
gtable	A table (vector or data frame) of values for selection.
gslider	A slider to select from a sequence values.
gspinbutton	A spinbutton to select from a sequence of values.
gedit	Single line of editable text.
gtext	Multiline text edit area.
ghtml	Display text marked up with HTML.
gdf	Data frame viewer and editor.
gtree	A display for hierarchical data.
gimage	A display for icons and images.
ggraphics	A widget containing a graphics device.
gsvg	A widget to display SVG files.
gfilebrowse	A widget to select a file or directory.
gcalendar	A widget to select a date.
gaction	A reusable definition of an action.
gmenubar	Add a menu bar to top-level window.
gtoolbar	Add a toolbar to a top-level window.
gstatusbar	Add a status bar to a top-level window.
gtooltip	Add a tooltip to a widget.
gseparator	A widget to display a horizontal or vertical line.

Table 2.5: Constructors for control widgets in gWidgets.

Constructor	Description
gwindow	Creates a top-level window.
ggroup	Creates a box-like container.
gframe	Creates a box container with a text label.
gexpandgroup	Creates a box container with a label and a trigger to expand/collapse.
glayout	Creates a grid container.
gpanedgroup	Creates a container for two child widgets with a handle to assign allocation of space.
gnotebook	Creates a tabbed notebook container for holding a collection of child widgets.

Table 2.6: Constructors for container objects.

And in table 2.7 we can see the main container methods used to do different functions on the widgets:

Method	Description
svalue	Ger or set widget's main property.
update	Update widget value.
show	Show widget if not visible.
font	Set a widget's font.
length	Length of data store.
names	Names of data store.
add	Adds a child object to a parent container called when a parent. container is specified to the container argument of the widget constructor, in which case the arguments are passed to this method.
delete	Removes a child object from a parent container.
dispose	Destroy widget or its parent.
enabled	Sets sensitivity of child components.
visible	Hides or shows object or part of object.

Table 2.7: Generic container methods provided by the gWidgets package.

2.6.3 EVENT HANDLERS

Event handlers activates different instructions when an event, such as a mouse click, related to some widget happens.

For example, if an event handler for a button is not defined, this button will not initiate any action. Hence we need to add an event handler to be called when an event occurs.

Let us explain this through a classic example, a **search button**. The **search button** is created with:

```
search_button <- gbutton("Search", cont = group)
```

When this button is clicked it will open a search menu. In this example our event is a button click, and the action we want consists in searching file names following a specific pattern and presenting the results. This action will be achieved as se can see in here,

```
addHandlerChanged(search_button,handler=function(h,...){
    pattern<- glob2rx(svalue(txt_pattern))
    file_names <- dir(svalue(start_dir), pattern,
                  recursive=TRUE)
    if (lenght(file_names))
          svalue(search_results) <- file_names
    else
    galert("No matching files found", parent = window)
})
```

The `addHandlerChanged` command applied on the **search button** starts several instructions when the button is pressed.

And in table 2.8 we can see the main event handlers:

Method	Description
addHandlerChanged	Primary handler call for when a widget's value is "changed". The interpretation of "change" depends on the widget.
addHandlerClicked	Set handler for when widget is clicked with (left) mouse button.
addHandlerDoubleclick	Set handler for when widget is double-clicked.
addHandlerRightclick	Set handler for when widget is right-clicked.
addHandlerMouseMotion	Set handler for when widget has mouse go over it.

Table 2.8: Generic functions to add callbacks in gWidgets package.

Chapter 3

USING R GUIs TO SOLVE A SENSITIVITY PROBLEM

Firstly, we are going to schematize the model. More details of the model could be consulted in the working paper annexed (see appendix D) at the final of this project. Secondly we will explain the different basic GUIs that have been created and their use. Finally we will show and explain the final GUI.

3.1 THE MODEL

The aim of the model is to simulate the reproductive cycles of the cows of a farm in order to study the sensitivity that will have different parameters over the final benefit.

To simulate the reproductive cycles of a cow many parameters has to be taken into account.

First of all the model assumes that the milk production over time, in each cycle of a cow, is well adjusted by the *Wood model*,

$$f(x) = Ax^B e^{-Cx}, \tag{3.1}$$

where parameters A, B and C depends on the cycle.

Parameters A, B and C depend on the cycle (see table 3.1): first, second or third. The rest of cycles have the same parameter values than the third one.

	1st	2nd	others
A	$A_1 \in \{8, \ldots, 24\}$	$A_2 = 1.54 \cdot A_1$	$A_3 = 1.47 \cdot A_1$
B	$B_1 = 0.208$	$B_2 = 0.179$	$B_3 = 0.209$
C	$C_1 = 0.002$	$C_2 = 0.003$	$C_3 = 0.0036$

Table 3.1: Coefficients of the Wood Model

In fact, the parameter A is the only one that is not fixed. This parameter determines how good is a cow. Remember that the more milk a cow gives the better the cow is. Not all the cows are equal, then there are considered 17 cow types, from 8 to 24. If a cow is from 8 it will not produce benefits to the farmer. And if a cow is from 20 it will be a very preciated cow because it will produce a lot of milk and benefits to the farmer.

In figure 3.1 we can see how varies the production of milk of a cow in each peridod depending on the parameter A.

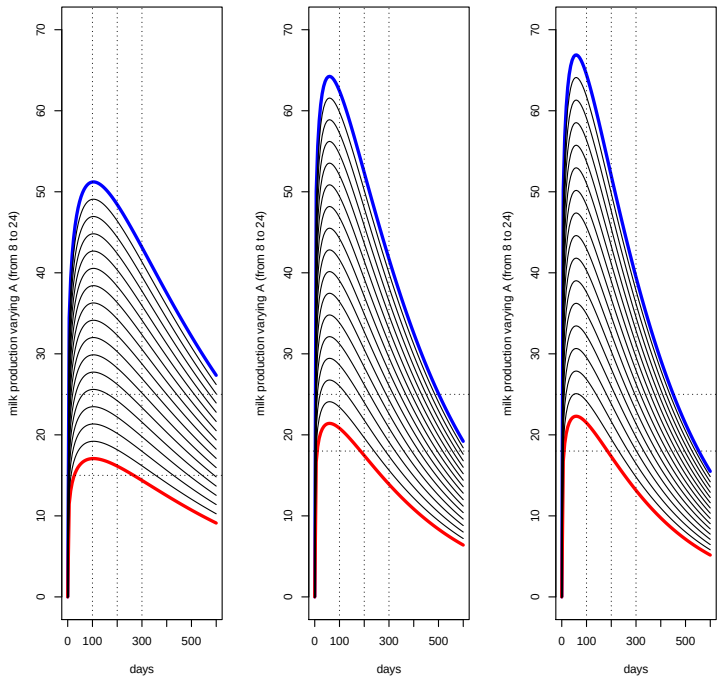

Figure 3.1: Variation of milk production depending on the parameter A.

In the previous figure 3.1 we differentiate three graphics, one for each period. In each graphic there are 17 curves, one for each value of A, from 8 to 24. The dashed vertical lines correspond to special dates and the horizontal ones to thresholds L_1, L_2 defined by *Trebol* protocol (see section 1.1).

There are many variables that has to be taken into account to analyze the whole behavior of a cow's evolution, and the model is defined by all this variables. We can differentiate three types of variables:

- **Basic random variables:** the random variables describe different important aspects of a cow. The most important of them will be explained with more detail in section 3.2.
- **Deterministic parameters:** these parameters are those whose values can be modified by farmers-decisions, economic reasons and others, but not in a random way.
- **New random variables:** these are random variables defined as functions of the previous ones.

The name of the variables and their definition can be consulted in the appendix D at the end of this project.

3.2 THE FOUR BASIC GUIs

There are four basic random variables:

- Non-forced death time
- Initiation of the reproductive cycle
- Number of unsuccessful inseminations
- Gestation time

For these random variables, a proper distribution has to be choosen. The probability law must reflect the real world. This is a difficult task because therefore the rules can be defined by the expertises there are no statistical studies about cow's liferules.

For example, a farmer knows that a 50% of the cows initiate its reproductive cycle before 60 days, or that the 10% of the cows die naturally in the first year and a 50% in 1000 days. Then, from that point on it has to be chosen a proper distribution that reflects what farmers know by their experience taking into account the sample size. Apparently different results could be obtained by simulation, when the sample size changes dramatically. Clearly the bigger will be the sample size the more accurate to the theorical distribution chosen it will be.

The four distributions were chosen by mathematicians. In order to see if this distributions are reliable we create four GUIs. These GUIs will allow to veterinarians to decide if the distributions are correct or not.

3.2.1 NON-FORCED DEATH TIME

This random variable describe the non-forced death time of a cow. The non-forced death time is the time it takes to the cow to die by natural causes, either by old age, illness, etc.

It follows a Weibull distribution. The Weibull's distribution function is

$$F(x;k,\lambda) = 1 - e^{(\frac{x}{\lambda})^k}\mathbf{1}_{[0,\infty]}(x), \tag{3.2}$$

and its density function is

$$f(x;k,\lambda) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(\frac{x}{\lambda})^k}\mathbf{1}_{[0,\infty]}(x), \tag{3.3}$$

where $k > 0$ is the shape parameter and $\lambda > 0$ is the scale parameter of the distribution.

The default values of the shape (a) and scale (b) parameters are:

$$a = 2, b = 1000(log2)^{-\frac{1}{a}} \tag{3.4}$$

This condition provides that a 10% of the cows will die during the first year and a 50% in 1000 days.

The GUI built for this variable allows the user to change different parameters, as the sample size and the shape. We can see it in figure 3.2.

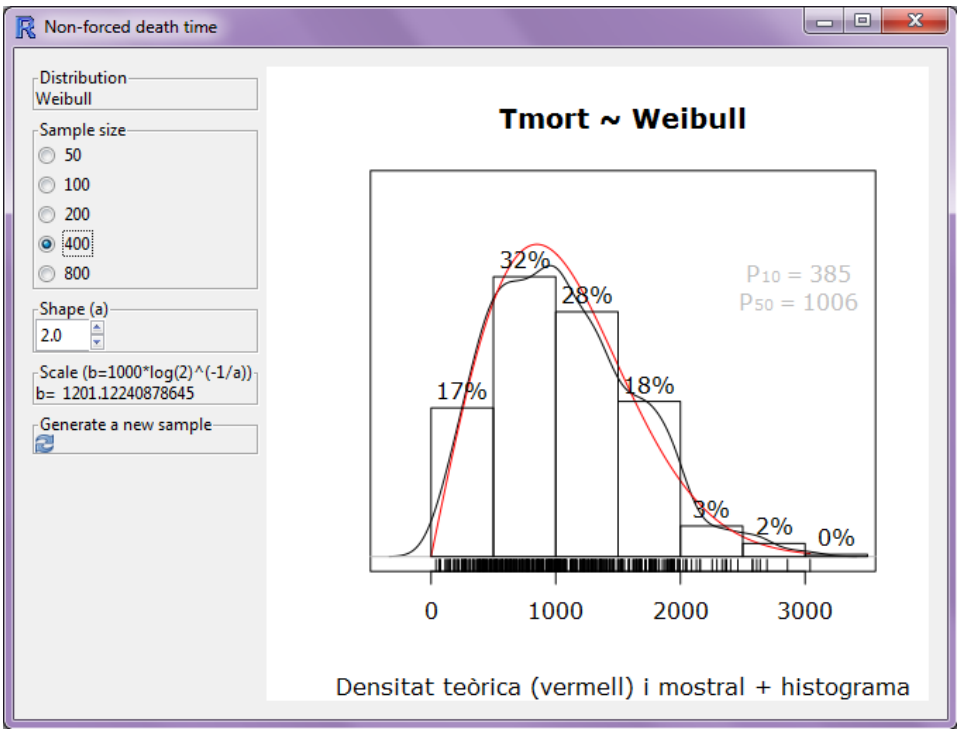

Figure 3.2: GUI to estimate the non-forced death time.

As we could see, the sample size is shown with radio buttons, because we wanted to see how vary the results depending on the farm size. A farm with 800 is considered a big farm while a farm of 50 cows is considered a little farm. Logically, the bigger the sample the more adjusted will be the results to the theorical distribution (red line). This is because in a farm of only 50 cows the variance is bigger than in a farm of hundreds of cows.

We could change also the shape (a) of the Weibull distribution using a spin button (see the section 2.4.2). We have decided using this type of widget here because it allows to increase or decrease the value of the shape slowly by 0.1 at the same time that the direct value could be writen in the blank. The value of the scale (b) is shown as a label and it changes every time we change the value of the shape because b is in function of a. This has been done with the `addHandlerChanged`control (see the section 2.6.3).

Finally there is an icon to refresh and generate a new sample of the same size to see how vary the results.

Note that the graphic shows different percentiles that reflects what we wanted. According to figure 3.2 10% of cows die during the first 385 days, and a 50% have died until the day 1006.

3.2.2 INITIATION OF THE REPRODUCTIVE CYCLE

We select the Weibull distribution to model the initiation of the reproductive cycles of a cow.

In section 3.2.1 we have defined the Weibull's distribution and density functions.

For the initiation of the reproductive cycle we choose a modified Weibull distribution with origin translated to 40 (because the cycle cannot start before the 40th day), with the following parameters a (shape) and b (scale) as default values:

$$a = 2, b = 20(log2)^{-\frac{1}{a}} \tag{3.5}$$

This condition provides that a 50% of the cows initiate the cycle before 60 days. All them are independent and identically distributed.

The Weibull distribution displaced (through an addicional parameter that we have called "Initial Position") has the following density function:

$$f(x;k,\lambda) = \frac{k}{\lambda}\left(\frac{x-\theta}{\lambda}\right)^{k-1} e^{-(\frac{x-\theta}{\lambda})^k} \mathbf{1}_{[\theta,\infty]}(x) \tag{3.6}$$

where $k > 0, \lambda > 0$ and θ determines the initial position of the distribution.

The GUI built for this variable allow the user to change different parameters, as the sample size, the initial position and the shape. We can see it in figure 3.3.

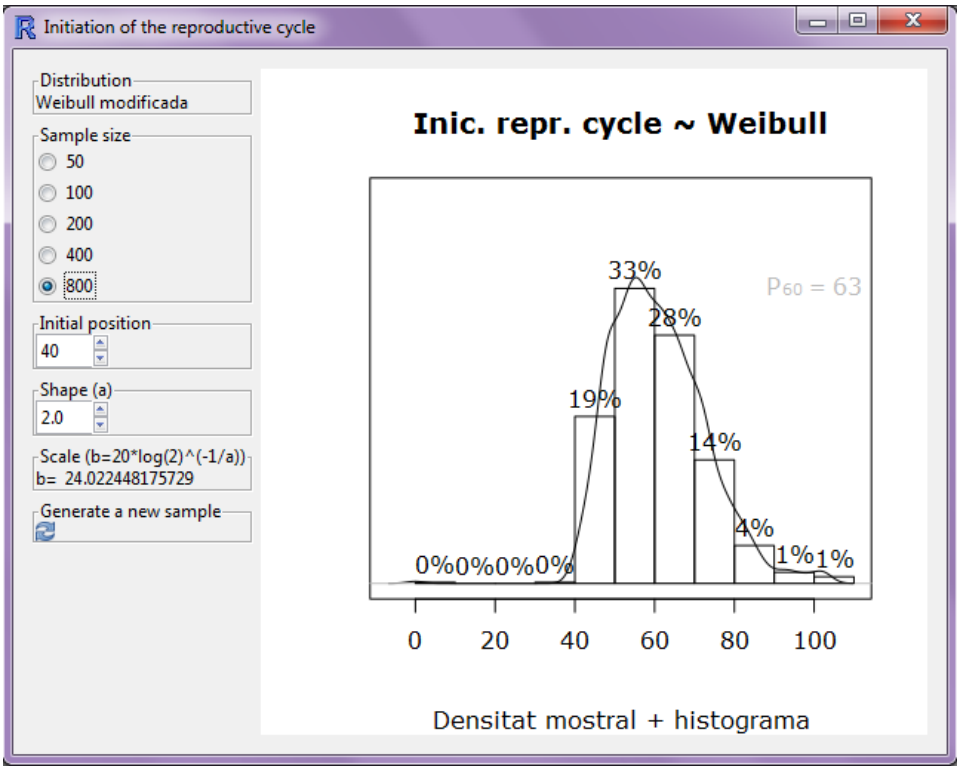

Figure 3.3: GUI to estimate the initiation of the reproductive cycle.

As before, the sample size could be changed through radio buttons. We could change also the initial position (θ) and the shape (a) of the modified Weibull distribution using spin buttons. The value of the scale (b) is shown as a label and it changes every time we change the value of the shape parameter a, according to 3.2.2.

Finally there is an icon to refresh and generate a new sample of the same size to see how vary the results.

Note that the graphic shows the percentile 60 that reflects what we wanted. According to figure 3.3, 50% of cows initiates its reproductive cycle during the first 63 days.

3.2.3 NUMBER OF UNSUCCESSFUL INSEMINATIONS

We consider that the number of unsuccessful inseminations in a cow before starting a new reproductive cycle. It follows a Geometric distribution.

The Geometric's probability function is

$$P(X = n) = q^{n-1}p, \forall n \geq 0 \tag{3.7}$$

It defines the probability that n tests will be necessary to obtain a successful insemination. It is a discrete distribution.

It only depends on the probability of successful insemination (p), whose default value is:

$$p = 0.15 \tag{3.8}$$

The GUI built for this variable allow the user to change different parameters, as the sample size and the probabibility of unsuccessful insemination. We can see it in figure 3.4.

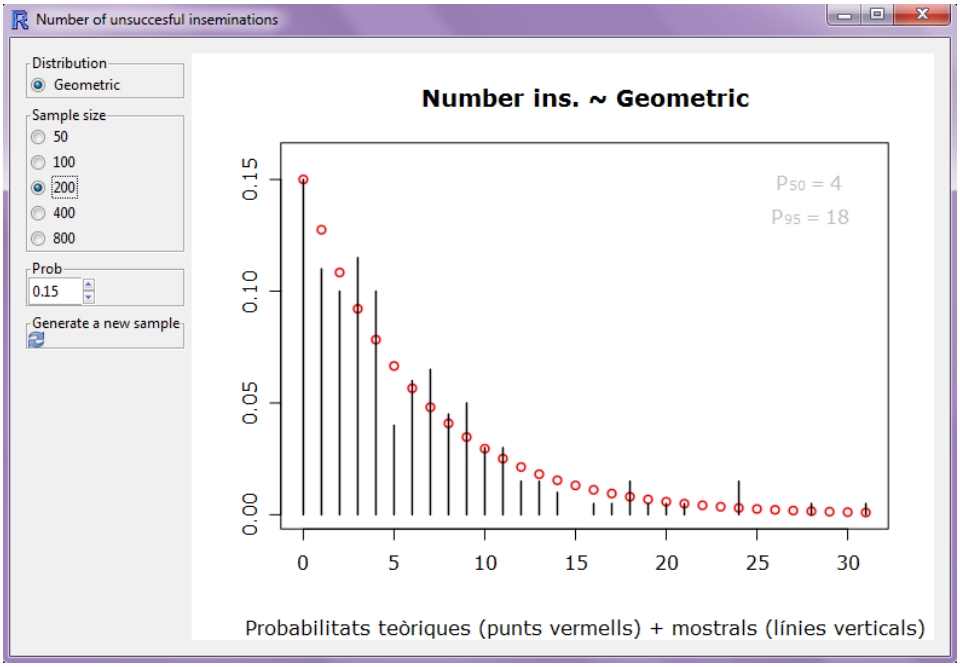

Figure 3.4: GUI to estimate the number of unsuccessful inseminations.

As the previous case, the sample size could be changed through radio buttons. We could change also the probability of an unsuccessful insemination (p) using a spin button.

Finally there is an icon to refresh and generate a new sample of the same size to see how vary the results.

Note that the graphic shows the percentile 50 and 95. According to figure 3.4 50% of cows are well-inseminated before 4 intents, and the 95% before 18 intents.

3.2.4 GESTATION TIME

We assume that the gestation time of a cow follows a Gaussian distribution. The Gaussian's probability density is:

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.9}$$

where σ is the standard deviation and μ the mean.

The default values of the standard deviation ($s.gest$) and mean ($m.gest$) parameters are:

$$m.gest = 280, s.gest = 3 \tag{3.10}$$

The GUI built for this variable allow the user to change different parameters, as the sample size, the mean and the deviation. We can see it in figure 3.5.

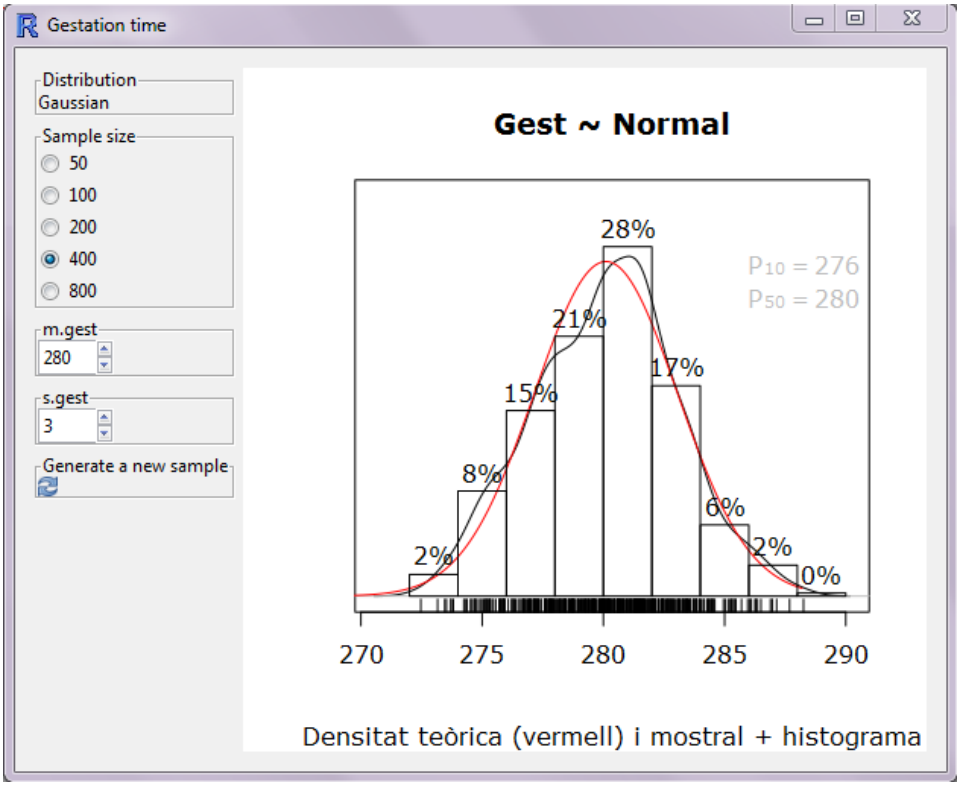

Figure 3.5: GUI to estimatethe the gestation time.

As in the other cases, the sample size could be changed through radio buttons. We could change also the mean ($m.gest$) and the standard deviation ($s.gest$) of the Gaussian distribution using spin buttons.

Finally there is an icon to refresh and generate a new sample of the same size to see how vary the results.

Note that the graphic shows the percentile 10 and 50. According to figure 3.3 10% of cows are pregnant 276 while a 50% are pregnant during 280 days.

3.3 PROGRAMMING THE FINAL GUI

3.3.1 DATA OF OUR GUI

The aim of the final GUI is to show the different values of the parameters used in the model.

It will be organized in three parts.

- **Simulation parameters:** Here will find all the parameters that are involved in the simulation of the cow: gestation's mean time, gestation's time deviation, probability that the meat will be marketable, drying for pregnancy, law that defines the arrival of cows to the farm, successful insemination probabibility and period between inseminations.
- **Price parameters:** Here there will be all the parameters that determine the price of the milk, the price of a cow and the cost of a cow.
- **Protocol parameters:** Here we show all the parameters that are taken into consideration to determine the dates and potential alarms.

3.3.2 WHICH WIDGETS USE

As we have seen in Chapter 2 there are many different widgets. It is very important to decide which widget is the more appropiate for each parameter.

There are fixed and variable parameters. Fixed parameters will be shown in labels, because even though they cannot be modified, they have to appear in the GUI as information of their value .

Variable parameters will be shown in different widgets depending on their function. For example, to determine the law that defines the arrival of cows to the farm we will use a radio button, because it shows the different options in a clear way but only one can be chosen.

To determine the number of simulations, the probability of insemination, the period between inseminations, and the different costs, spinbuttons will be used. These buttons will allow the user to increase or decrease the values of the different parameters as well as to write a new value in the space provided for each parameter.

Finally, combo boxes will be used to fix the price of the milk, the price of buying a cow, and the price of selling one. This combo boxes will be very

useful because, despite being similar to radio buttons, they display only the value currently selected, which saves space in the GUI.

And last but not least there will be a widget that contains the action of refreshing, which generates a new sample. This widget could be a button or an icon.

3.3.3 FINAL GUI

The final GUI has to implement the final function proposed by the mathematicians and show the results of this function. At the same time it has to bring the possibility of modifying manually all the possible parameters that are susceptible to change.

As we have said, the final GUI is sectioned into three parts, one for each group of parameters. There is also a space to show the graphic result. In Figures 3.7 and 3.9 we could see the aspect of the final GUI with 1000 and 10000 simulations respectively.

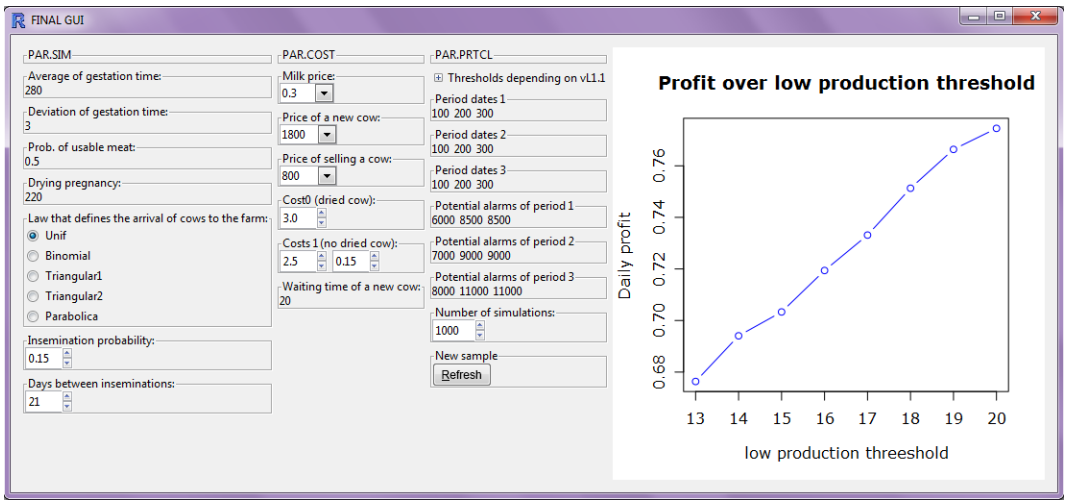

Figure 3.6: Final GUI with 1000 simulations.

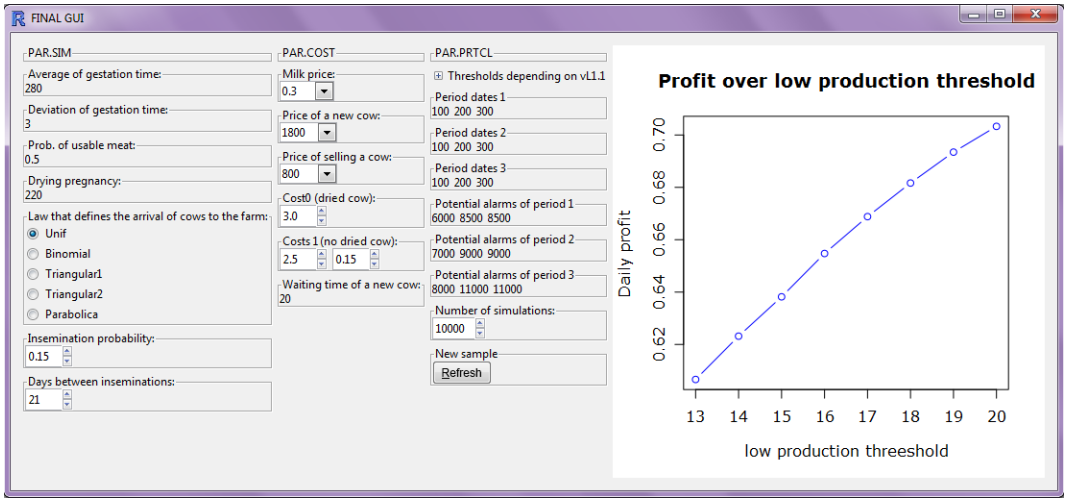

Figure 3.7: Final GUI with 10000 simulations.

We have used labels to show different parameters that are important but that cannot be changed by the interface.

In the first section of the interface, `PAR.SIM`, we have:

- The **average and deviation of gestation time** are fixed parameters with values of 280 and 3 days respectively. It means that usually a cow is pregnant between 277 and 283 days with a probability of about 2/3, being 280 days the average. These values were decided by the experts as the most realistic. Since these parameters are fixed, they are shown in labels.

- The **probability of marketable meat** is the probability that the meat could be sold once the cow is dead. As we can see there is a probability of 50% that the meat could be profitable. This datum is given by the experts in the cows field and for the moment it is fixed.

- The **drying pregnancy** is the interval after initiating pregnancy for the "biological drying", with 220 days as the default value. By the moment this parameter is fixed and it is shown in a label.

- The **law that defines the quality of cows that arrives to the farm** is the law that defines the parameter A on the Wood model of the cows that arrive to the farm (see section 3.1). There are different laws to choose from to study how the final benefit varies depending on them. Right now, the experts believe that the arrival of cows to the farm follows a uniform distribution. This means that there are the equal chances that the farm get a cow of type 10 ($A = 10$) than of type 24 ($A = 24$). To assess whether the likelihoods that a cow be of one type or another are really equal, different laws has been added to the final GUI in order to see which one fits best the farms reality.

In figure 3.8 we can see the arrival of cows of different types (diferent A values) following a uniform distribution against different laws.

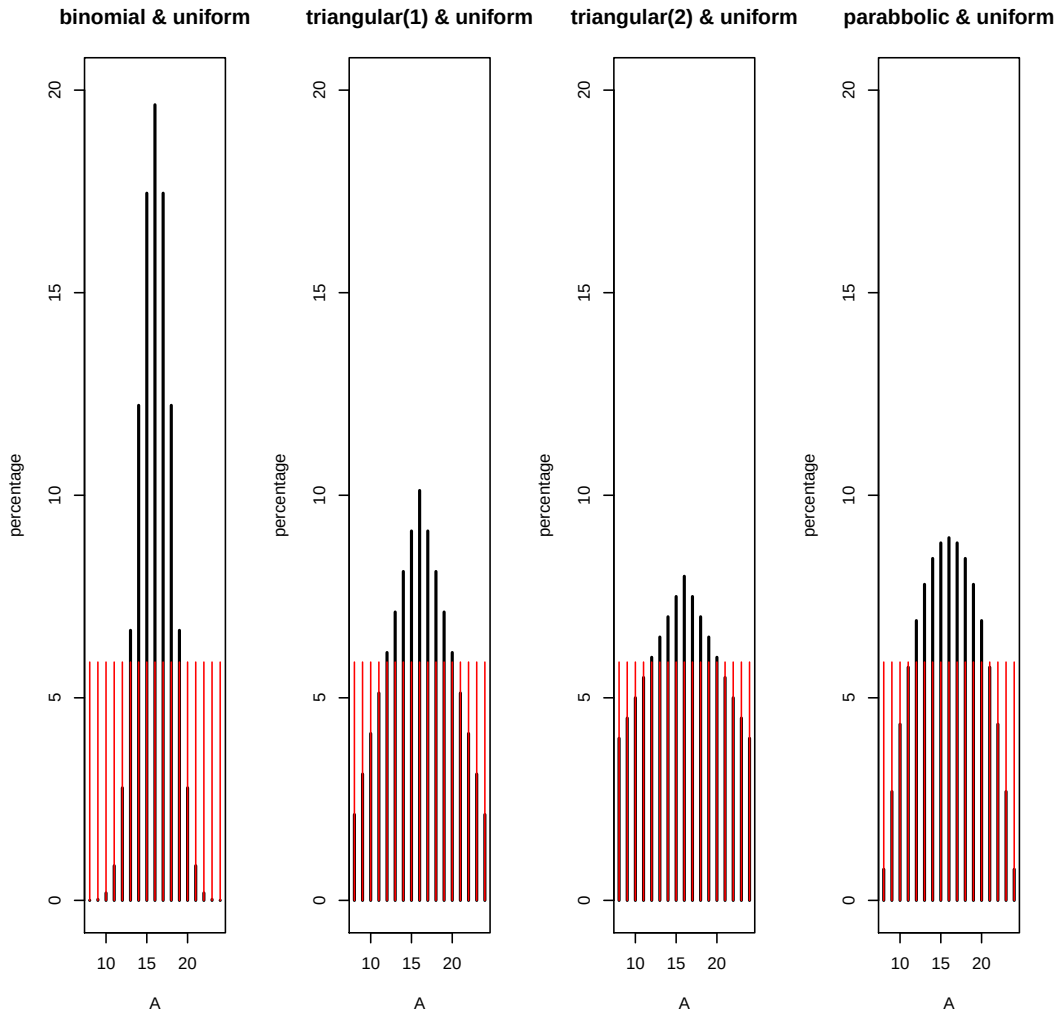

Figure 3.8: Several farms compositions.

- The **successful insemination probability** is not fixed, because it may depend on the farm. For this reason it is shown as a spin button.
- The **days between inseminations** are the days that the farmer will wait between unsuccessful consecutive inseminations of the cow. As in the previous case, it could also change depending on the farm and its protocol.

In the second section of the interface, `PAR.COST`, we have:

- The **milk price** per liter. This is the price that the farmer will earn for selling the milk. Obviously it can change depending on the economy or

farmer decisions, so it is shown as a combo box, because it allows the farm to choose one price of the list or write a different price in the blank.

- The **price of a new cow** is what it has to be paid to acquire a new cow. It is also a variable parameter so it is shown too in a combo box.

- The **price of selling a cow** is what will be earned for selling a cow. As before it can vary so it is shown in a combo box.

- The **cost 0** is the constant daily cost of a cow in its dry period, being 3 euros the default value. It may vary but not much, hence it is shown as a spin button.

- The **cost 1** is the constant daily cost of a cow in its productive period. It is formed by a fixed and a variable cost. The fixed daily cost of a productive cow is defined by cost1.1, being 2.5 euros the default value, and the variable daily cost is calculated by multiplying the cost1.2 by the liters of milk produced, with 0.15 euros as a default value. It may vary but not much, so it is shown also as a spin button.

- The **waiting time of a new cow** is then number of days that a farm has to wait to acquire a new cow when a cow has died unexpectedly. This value is fixed at 20 by experts, so it is shown as a label.

And finally in the third section of the interface, `PAR.PRTCL`, all parameters are shown as labels because they are fixed parameters defined by the "trebol" protocol. We have:

- The **thresholds depending on vL1.1** are the two thresholds defined on the model depending on the first component of the first threshold (vL1.1), i.e., it shows the values of threshold1 and threshold2 for differents values of vL1.1 (see section 1.1). This is not necessary to be always displayed, and for this reason we have used a gexpandgroup to show it. It can be seen when the small cross will be clicked.

- The **period dates** are the period dates considered in the protocol of a farm. In our case the three period dates are the same.

- The **potential alarms of each period** are defined by each farm protocol. This potential alarms determine when a cow will be considered not profitable.

Finally, we have two extra widgets in our GUI. They are:

- The **number of simulations**, that is, the number of cows that we want to simulate in the farm. It is shown as a spinbutton but one can write directly the number of simulations that we want in the blank space.

- The **new sample** button. It is the refresh button, and generates a new sample with the same number of simulations and the same values for the parameters when it is pressed.

In Figure 3.9 we can see the two graphics obtained from the previous simulations, in which we can see that the daily benefit increases when the threshold L grows.

Within this simple model, it seems that the low production threshold can be increased to provide greater profit.

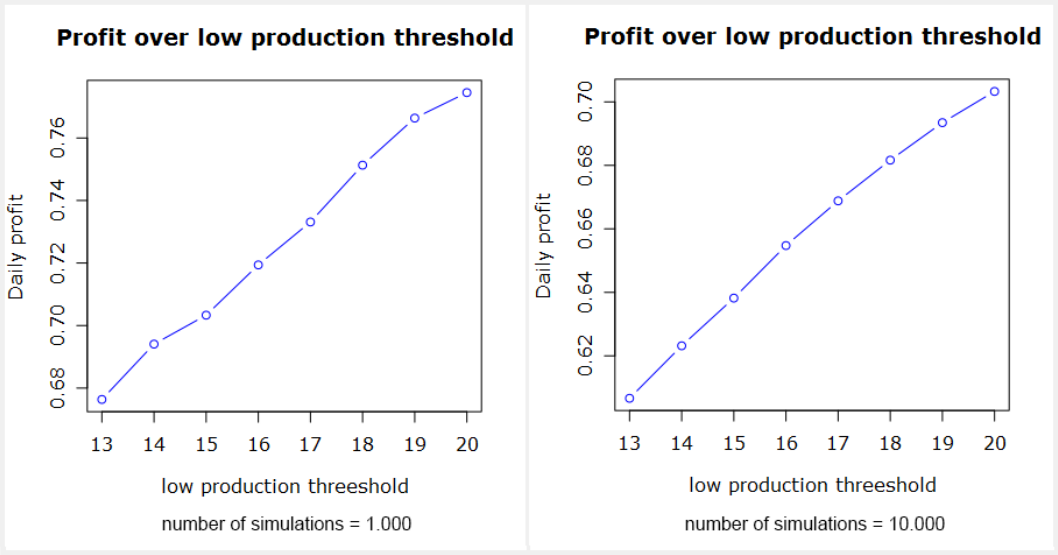

Figure 3.9: Graphics of the low production over threshold with 1000 and 10000 simulations respectively.

3.4 POSSIBLE IMPROVEMENTS

This project has not finished here. Despite the GUI works well and do what we want, there are several things that could be improved in the future, namely:

- Reducing the execution time consumed by the GUI.
- Programming it with C and R jointly, because C is much faster for the computationally intensive parts. The package `Rcpp`provides a way to include `C++`code inside an R program.
- Trying to use the GUI always with the same random stream, depending on the sample size. This will permit a more efficient statistical comparison among different parameter sets.

Chapter 4

CONCLUSIONS

4.1 VALORATION

Visualization tools that allow to operate and interact with values' parameters are essential in the process of modeling and simulation.

In this work we have created and used GUIs for modeling specific sections, in order to assess if the laws of certain random variables are adjusted to reality. As well as we have created a GUI with a stochastic and highly complex multiparametrical function.

The main result of this work is the creation of four partial and one final interface. In fact, the partial interfaces are simpler and have served to facilitate the construction of the final GUI. Therefore, the project lies in the field of programming with R. However, do not forget that there is a stochastic modeling problem of a dairy farm behind. So, the project has required a dive in areas that are not programming, as distributional models, profit optimization, etc.

The goal behind it is to analyze the sensitivity of the daily profit function in relation to some of the parameters involved in the model (characteristics of reproductive cycles, pricing and protocols mainly) and that intend to reflect the reality of an operation of this type.

At this point, this project is not about saying which protocol is best, but to validate the model itself. However, if we interpret the result on the final figure 3.9 as definitive we would say that the benefit increases when the threshold L grows.

Now we have the tool from which in the future we could do statistical studies to quantify the effect that different parameters have on the final daily benefit of a dairy farm.

4.2 WHY THIS PROJECT?

I wanted to do a project in which I had to use applied Mathematics to solve a real lifeproblem.

When Mercè and Aureli proposed me to take part of this project doing the graphical part to study a model that it has to be used for real farmers I thought that it was a very good oportunity to develop different skills acquired as a Mathematician in a real problem.

With this project I had to learn from scratch how to create graphical user interfaces for R, and I had to search useful material to do it.

I think that it has been very useful to learn how to create different types of graphic user interfaces because they are a useful way to show the results of any work you create to other people. It's a very graphic way also of interacting with different functions and seeing then how they work. And one of their main advantages is that people don't need to know the code with which the GUI has been created to use it. Then anyone could use a GUI once it has been created.

Once you has learnt how to program a GUI with one language I think that is easier to learn to do the same in another language. It's for that too that I'm proud of what I've learnt during this project, and I think that probably in the future I will have the necessity of create GUIs to show mathematical results to people that are not mathematicians.

4.3 AKNOWLEDGMENTS

Firstly, I want to give thanks to my two tutors for all their patience and their support in what I did.

Secondly also show gratitude for the interest that the veterinarian Lorena Castillejos has shown in my work and her cooperation testing the GUIs that I have created.

Appendix A

GLOSSARY

Bindings links, something that binds. We use it to talk about links between programation languages.

Commands A signal that initiates an operation defined by an instruction.

Controls Set of instruments used to operate or regulate different widgets.

Dairy farming Dairy farming is a class of agricultural, or an animal husbandry, enterprise, for long-term production of milk, usually from dairy cows but also from goats, sheep and camels, which may be either processed on-site or transported to a dairy factory for processing and eventual retail sale. In this project we talk about a dairy farm from dairy cows.

R It is a freely available language and environment for statistical computing and graphics.

Threshold The point that must be exceeded to begin producing a given effect or result or to elicit a response. We talk about thresholds in the milk production.

Toolkit Software designed to perform a specific function to solve a problem.

Trebol protocol Is the name of the farm protocol in which we will focus.

Widget Any small mechanism or device, the name of which is unknown or temporarily forgotten.

Window A rectangular area on the screen that displays its own file or message independently of the other areas of the screen.

Wrapper That encompasses something.

Appendix B

SIMPLE WIDGETS EXAMPLES FROM GWIDGETS PACKAGE

In this appendix we will find graphical examples of the widgets used in gwidgets package. We can see that there are three examples of every widget, because of depending on the operating system one works, the widgets has diferent aspects.

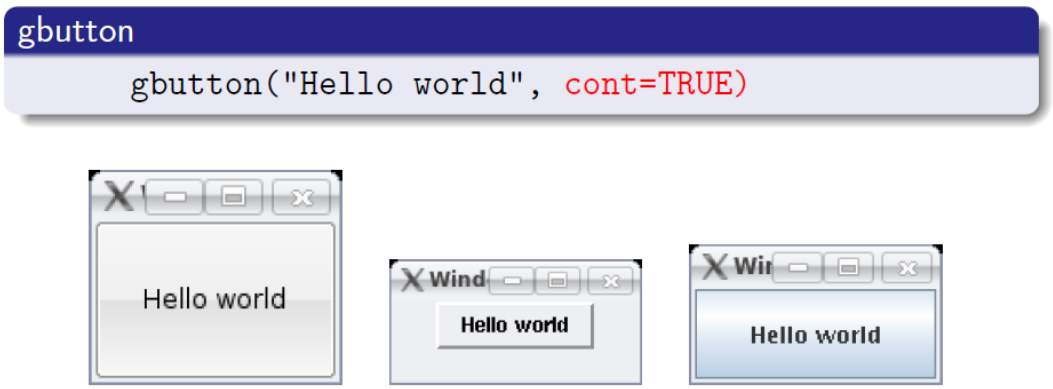

Figure B.1: A gbuton widget.

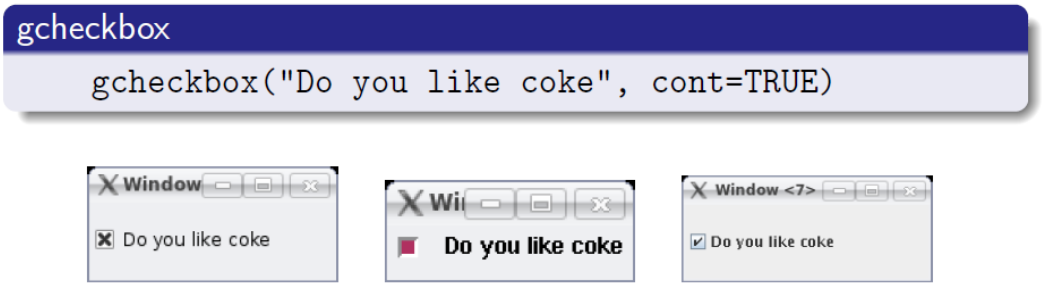

Figure B.2: A gcheckbox widget.

gcheckboxgroup

```
gcheckboxgroup(items, cont=TRUE)
```

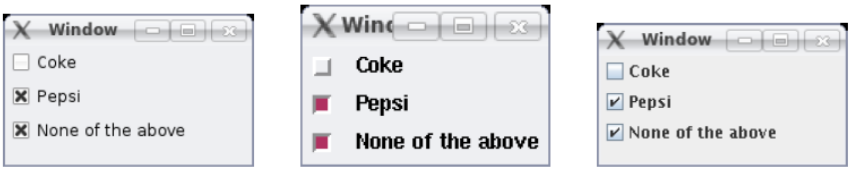

Figure B.3: A gcheckboxgroup widget.

gradio

```
items = c("Coke","Pepsi","None of the above")
gradio(items, cont=TRUE)
```

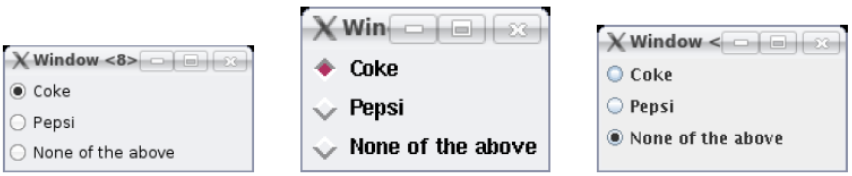

Figure B.4: A gradio buttons widget.

gdroplist

```
gdroplist(items, cont=TRUE)
```

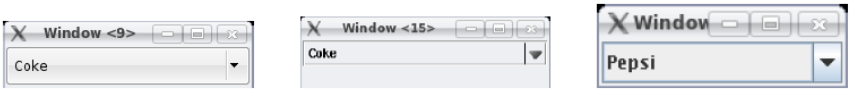

Figure B.5: A gdroplist widget.

gdroplist: aka a combobox

```
gdroplist(items, editable=TRUE, cont=TRUE)
```

Figure B.6: A gdroplist and combo box widget.

gedit

```
gedit("This space for rent", cont=TRUE)
```

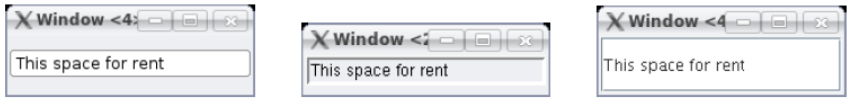

Figure B.7: A gedit widget.

gexpandgroup

```
win = gwindow("gexpandgroup")
g = gexpandgroup("Click to toggle", cont=win)
gbutton("click me", cont=g)
visible(g) <- TRUE  ## open up
```

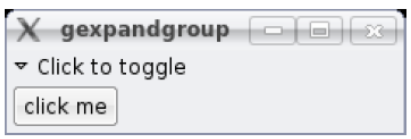

Figure B.8: A gexpandgroup widget.

gframe

```
win = gwindow("gframe")
gp = ggroup(cont=win)
g = gframe("Text label", cont=gp)
gbutton("click me", cont=g)
```

Figure B.9: A gframe widget.

glabel

```
glabel("Hello world", cont=TRUE)
```

Figure B.10: A glabel widget.

gpanedgroup

```
win = gwindow("gpanedgroup")
pg = gpanedgroup(cont=win)
gbutton("button 1", cont=pg)
gbutton("button 2", cont=pg) ## add twice
```

Figure B.11: A gpanedgroup widget.

gslider

```
gslider(from=0, to = 100, by = 1, cont=TRUE)
```

Figure B.12: A gslider widget.

gspinbutton

```
gspinbutton(from=0, to = 1, by = 0.1, cont=TRUE)
```

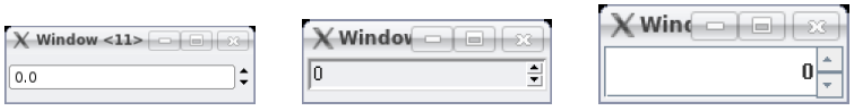

Figure B.13: A gspinbutton widget.

gdf

```
require(MASS)
gdf(Cars93, cont=TRUE)
```

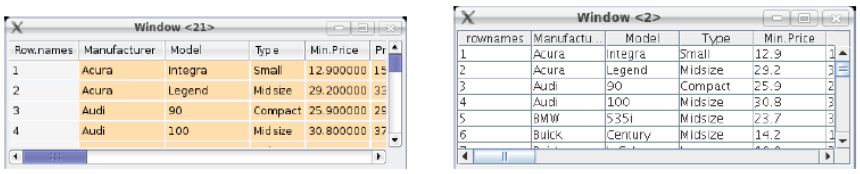

Figure B.14: A gdataframe widget.

```
gtable
gtable(items, multiple=TRUE, cont=TRUE)
```

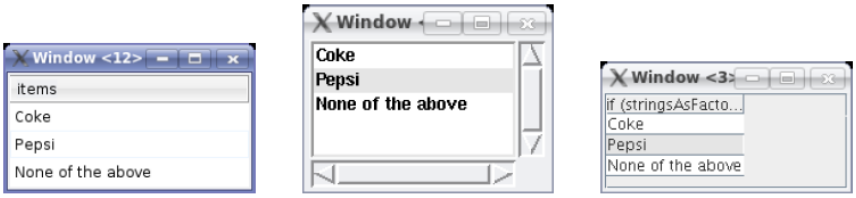

Figure B.15: A gtable widget.

```
gtable
gtable(mtcars,chosencol=6, cont=TRUE)
```

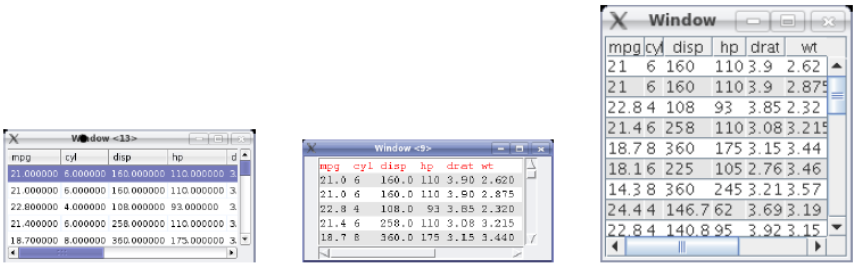

Figure B.16: Another gdataframe widget.

```
gtext
gtext("This space for rent",  cont=TRUE)
```

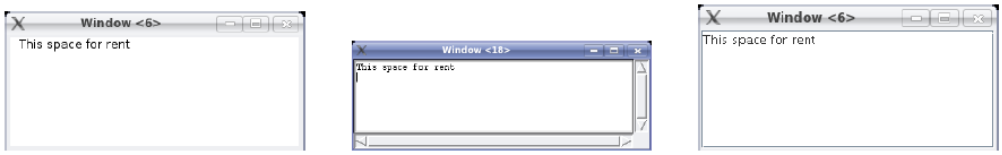

Figure B.17: A gtext widget.

gtoolbar

```
f = function(h,...) print("Hello world")
  tblst=list(
    open = list(handler=f, icon="open"),
    new  = list(handler=f, icon="new"),
    quit = list(handler=f, icon="close")
  )
gtoolbar(tblst, cont=TRUE)
```

Figure B.18: A gtoolbar widget.

gvarbrowser

```
gvarbrowser(cont=TRUE)
```

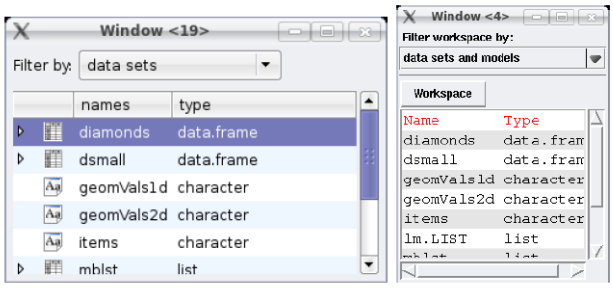

Figure B.19: A gvarbrowser widget.

button with icon

```
gbutton("ok", cont=TRUE)
```

Figure B.20: A button with icon widget.

ggraphics

```
> ggraphics(cont=TRUE)
> hist(rnorm(100))
> dev.list()
Cairo
    2
```

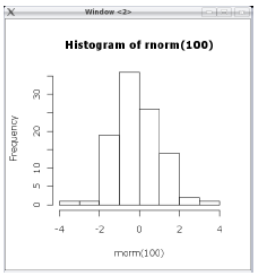

Figure B.21: A ggraphics widget.

gimage

```
png("/tmp/rnorm.png")
hist(rnorm(100))
dev.off()  ## tcltk has limited number of formats
system("convert /tmp/rnorm.png /tmp/rnorm.gif")
gimage("/tmp/rnorm.gif", cont=TRUE)
```

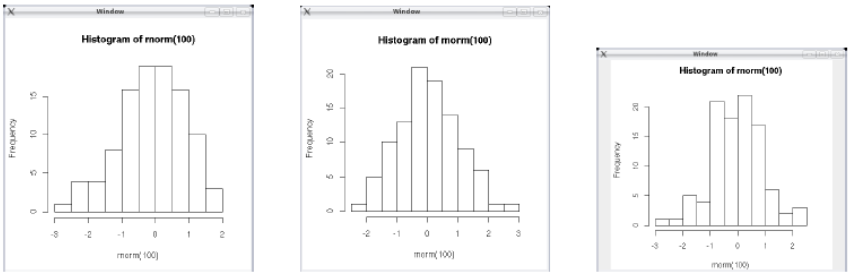

Figure B.22: A gimage widget.

gmenu

```
mblst = list(
  File=list(
    open = list(handler=f,icon="open"),
    quit = list(handler=f,icon="close")
  ),
  Edit = list(
    cut  = list(handler=f),
    copy = list(handler=f)
  )
)
gmenu(mblst, cont=TRUE)
```

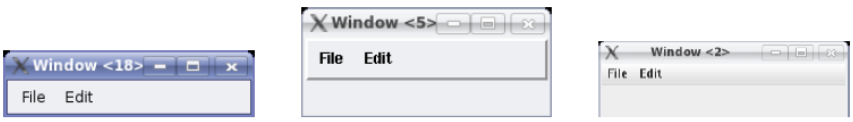

Figure B.23: A gmenu widget.

gnotebook

```
win = gwindow("gnotebook")
nb = gnotebook(cont=win)
gbutton("button",   cont=nb, label = "tab label")
gbutton("button 2", cont=nb, label = "tab label 2")
```

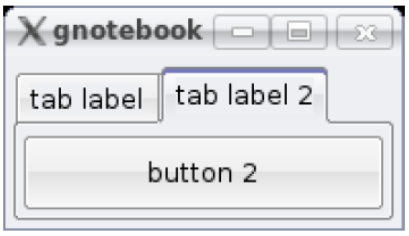

Figure B.24: A gnotebook widget.

Button box (take 1)

```
win = gwindow("Button box")
g = ggroup(cont = win)
gbutton("cancel", cont=g)
gbutton("ok", cont=g)
```

Figure B.25: Button box simple.

Button box (take 2)

```
win = gwindow("Button box")
g = ggroup(cont = win)
gbutton("cancel", cont=g)
addSpring(g)
gbutton("ok", cont=g)
```

Figure B.26: Button box aligned.

Button box (take 3)

```
win = gwindow("Button box")
g = ggroup(cont = win)
gbutton("cancel", cont=g, expand=TRUE)
gbutton("ok", cont=g)
```

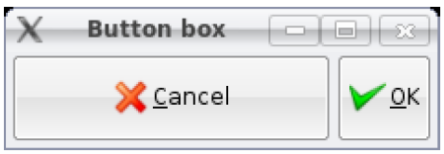

Figure B.27: Button box with big buttons.

Appendix C

THE CODES

C.1 Librarys used

First of all we have to load this packages to create all the GUI's:

```
library(gWidgets)
library(gWidgetsRGtk2)
library(cairoDevice)
library(lattice)
```

Table C.1: Necessary librarys.

C.2 Codes of the partial GUIs to determine the basic random variables

For the next four GUI's we will use a function to create histograms with a percentage on the bottom of each bar. The function is defined below (table C.2):

```
histog<- function(x, ...){
     H <- hist(x,plot=FALSE)
     H$density <- with(H, 100*density* diff(breaks)[1])
     abs <- paste(round(H$density), "%", sep="")
     hist(x,freq=F,labels=labs,main="",xlab="",ylab="",axes=F,...)
}
```

Table C.2: Function to create the histograms with the percentage.

Here we have the code to create the 4 GUI's:

```
updatePlot<-function(h,...){
     x<-c(0,svalue(pos)+floor(rweibull(svalue(sampleSize),
     svalue(shape), 20*log(2)^(-1/(svalue(shape))))))
     y<-density(x)$y
     plot(density(x, adjust = 0.8,kernel = "gaussian"),
     ylim=c(0,max(y)*1.3), yaxt='n', ann=FALSE)
     title( main="Inic.  repr.  cycle W̃eibull",
     sub="Densitat mostral + histograma")
     histog(x,add=T)
     text(0.97*max(x),0.99*max(y), bquote(paste(P,
     scriptstyle(60))==.(round(quantile(x,0.6)))),col="gray")
}
distribution <- glabel("Weibull modificada", horizontal=FALSE,
     handler=NULL)
sampleSize <- gradio(c(50,100,200,400,800), handler = updatePlot)
shape<-gspinbutton(from=0.5,to=10,by=0.5, value=2, handler=updatePlot)
scale<-glabel(text="b=", editable=FALSE, handler=NULL)
scalevalue<-glabel("24.02244818",handler=NULL)
refresh<-gimage("refresh", dirname="stock", handler = updatePlot)
pos<-gspinbutton(from=20, to=60, by=1, value=40, handler=updatePlot)
window <- gwindow("Initiation of the reproductive cycle")
BigGroup <- ggroup(cont=window)
group <- ggroup(horizontal=FALSE, container=BigGroup)
 tmp <- gframe("Distribution", container=group)
 add(tmp, distribution)
 tmp <- gframe("Sample size", container=group)
 add(tmp,sampleSize)
 tmp <- gframe("Initial position", container=group)
 add(tmp,pos)
 tmp <- gframe("Shape (a)", container=group)
 add(tmp,shape)
 tmp <- gframe("Scale (b=20*log(2)^(-1/a))", container=group)
 add(tmp,scale)
 add(tmp,scalevalue)
 addHandlerChanged(shape, handler =function(h,...){
     svalue(h$action)<-svalue(20*log(2)^(-1/(svalue(shape))))
     }, action = scalevalue)
 tmp <- gframe("Generate a new sample", container=group)
 add(tmp, refresh)
add(BigGroup, ggraphics())
}
```

Table C.3: Modified Weibull function to estimate the initiation of the reproductive cycle.

```
availDists<- c(Geometric="rgeom")
updatePlot<-function(h,...){
     x<-do.call(availDists[svalue(distribution)],
     list(svalue(sampleSize),svalue(p)))
     t<-0:max(x)
     tab<-prop.table(prop.table(table(x)))
     dfx<-as.data.frame(tab)
     valors<-as.numeric(levels(dfx$x))
     f.rel<-dfx$Freq
     ymax<-max(f.rel,dgeom(t,svalue(p)))
     plot(t,dgeom(t,svalue(p)),type="p",col="red",lwd=1.5,
     ylim=c(0,ymax+.01),xlab="",ylab="")
     points(valors,f.rel,type="h", lwd=1.5)
     title( main="Number ins.  Ẽometric",
     sub="Probabilitats teòriques (punts vermell) +
     mostral (línies verticals)")
     text(0.9*max(x),0.99*ymax, bquote(paste(P,
     scriptstyle(50))==.(round(median(x)))),col="gray")
     text(0.9*max(x),0.89*ymax, bquote(paste(P,
     scriptstyle(95))==.(round(quantile(x,0.95)))),col="gray")
}
distribution <- gradio(names(availDists), horizontal=FALSE,
     handler=NULL)
sampleSize <- gradio(c(50,100,200,400,800), handler = updatePlot)
p<-gspinbutton(from=0,to=100,by=0.05,value=0.15,handler=updatePlot)
refresh<-gimage("refresh", dirname="stock", handler = updatePlot)
window <- gwindow("Initiation of the reproductive cycle")
BigGroup <- ggroup(cont=window)
group <- ggroup(horizontal=FALSE, container=BigGroup)
 tmp <- gframe("Distribution", container=group)
 add(tmp, distribution)
 tmp <- gframe("Sample size", container=group)
 add(tmp,sampleSize)
 tmp <- gframe("Probability", container=group)
 add(tmp,p)
 tmp <- gframe("Generate a new sample", container=group)
 add(tmp, refresh)
add(BigGroup, ggraphics())
}
```

Table C.4: Geometric function to estimate the number of unsuccessful inseminations.

```
updatePlot<-function(h,...){
     x<-do.call(availDists[svalue(distribution)],list(svalue(sampleSize),
     svalue(m.gest),svalue(s.gest)))
     y<-density(x)$y
     plot(density(x, adjust = 0.8,kernel = "gaussian"),
     ylim=c(0,max(y)*1.2), yaxt='n', ann=FALSE)
     title( main="Gest Ñormal",
     sub="Densitat teòrica (vermell) i mostral + histograma")
     t<-seq(0,max(x),by=0.1)
     lines(t,dnorm(t,mean=mean(x), sd=sd(x)),type="l",cex=0.8,col="red")
     histog(x,add=T)
     rug(x)
     text(max(x),0.99*max(y), bquote(paste(P,
     scriptstyle(10))==.(round(quantile(x,0.1)))),col="gray")
     text(max(x),0.89*max(y), bquote(paste(P,
     scriptstyle(50))==.(round(median(x)))),col="gray")
}
distribution <- glabel(names(availDists), horizontal=FALSE,
     handler=NULL)
sampleSize <- gradio(c(50,100,200,400,800), handler = updatePlot)
m.gest<-gspinbutton(from=0,to=500,by=1, value=280, handler=updatePlot)
s.gest<-gspinbutton(from=0,to=50,by=1, value=3, handler=updatePlot)
refresh<-gimage("refresh", dirname="stock", handler = updatePlot)
window <- gwindow("Initiation of the reproductive cycle")
BigGroup <- ggroup(cont=window)
group <- ggroup(horizontal=FALSE, container=BigGroup)
 tmp <- gframe("Distribution", container=group)
 add(tmp, distribution)
 tmp <- gframe("Sample size", container=group)
 add(tmp,sampleSize)
 tmp <- gframe("m.gest", container=group)
 add(tmp,m.gest)
 tmp <- gframe("s.gest", container=group)
 add(tmp,s.gest)
 tmp <- gframe("Generate a new sample", container=group)
 add(tmp, refresh)
add(BigGroup, ggraphics())
}
```

Table C.5: Gaussian function to estimate the gestation time.

```
updatePlot<-function(h,...){
     x<-do.call(availDists[svalue(distribution)],list(svalue(sampleSize),
     svalue(shape),1000*log(2)^(-1/(svalue(shape)))))
     y<-density(x)$y
     plot(density(x, adjust = 0.8,kernel = "gaussian"),
     ylim=c(0,max(y)*1.3), yaxt='n', ann=FALSE)
     title( main="Tmort   Weibull",
     sub="Densitat teòrica (vermell) i mostral + histograma")
     t<-seq(0,max(x),by=0.1)
     lines(t,dweibull(t,svalue(shape),1000*log(2)^(-1/(svalue(shape)))),
     type="l",col="red")
     histog(x,add=T)
     rug(x)
     text(0.97*max(x),0.99*max(y), bquote(paste(P,
     scriptstyle(10))==.(round(quantile(x,0.1)))),col="gray")
     text(0.97*max(x),0.89*max(y), bquote(paste(P,
     scriptstyle(50))==.(round(median(x)))),col="gray")
}
distribution <- glabel(names(availDists), horizontal=FALSE,
     handler=NULL)
sampleSize <- gradio(c(50,100,200,400,800), handler = updatePlot)
shape<-gspinbutton(from=0.5,to=10,by=0.5, value=2, handler=updatePlot)
scale<-glabel(text="b=", editable=FALSE, handler=NULL)
scalevalue<-glabel("1201.122408",handler=NULL)
refresh<-gimage("refresh", dirname="stock", handler = updatePlot)
window <- gwindow("Initiation of the reproductive cycle")
BigGroup <- ggroup(cont=window)
group <- ggroup(horizontal=FALSE, container=BigGroup)
 tmp <- gframe("Distribution", container=group)
 add(tmp, distribution)
 tmp <- gframe("Sample size", container=group)
 add(tmp,sampleSize)
 tmp <- gframe("Initial position", container=group)
 add(tmp,pos)
 tmp <- gframe("Shape (a)", container=group)
 add(tmp,shape)
 tmp <- gframe("Scale (b=1000*log(2)^(-1/a))", container=group)
 add(tmp,scale)
 add(tmp,scalevalue)
 addHandlerChanged(shape, handler =function(h,...){
    svalue(h$action)<-svalue(1000*log(2)^(-1/(svalue(shape))))
    }, action = scalevalue)
 tmp <- gframe("Generate a new sample", container=group)
 add(tmp, refresh)
add(BigGroup, ggraphics())
}
```

Table C.6: Weibull function to estimate the non-forced death time

C.3 Code of the final GUI

```
############################
######### 1a part #########
############################


# Aquest codi s'ha d'executar de manera automatica nomes
#"1 vegada" a l'iniciar la sessio


###------------------------- Wood

f<-function(x,A,B,C){A*x^B*exp(-C*x)}


###------------------------- Punts de tall

arrels <-function(A=16,L1=c(15,18,18),L2=c(25,25,25),
dates=c(100,200,300,100,200,300,100,200,300) )
{
  A1<-A
  B1<-0.208
  C1<-0.002
  A2<-A1*1.54
  B2<-0.179
  C2<-0.003
  A3<-A1*1.47
  B3<-0.209
  C3<-0.0036
### LLindar L2=25 per defecte
if (f((B1/C1),A1,B1,C1) <= L2[1]){l2.1<-dates[1]}
if (f((B1/C1),A1,B1,C1) > L2[1]){
  l2.1<-floor(uniroot(function(x)
  {f(x,A=A1,B=B1,C=C1)-L2[1]},c(ceiling(B1/C1),2000),
  tol=1e-10)$root)
}
if (f((B2/C2),A2,B2,C2) <=  L2[2]){l2.2<-dates[4]}
if (f((B2/C2),A2,B2,C2) > L2[2]){
  l2.2<-floor(uniroot(function(x)
  {f(x,A=A2,B=B2,C=C2)-L2[2]},c(ceiling(B2/C2),2000),
  tol=1e-10)$root)
}
if (f((B3/C3),A3,B3,C3) <=  L2[3]){l2.3<-dates[7]}
if (f((B3/C3),A3,B3,C3) > L2[3]){
  l2.3<-floor(uniroot(function(x)
  {f(x,A=A3,B=B3,C=C3)-L2[3]},c(ceiling(B3/C3),2000),
```

```
  tol=1e-10)$root)
}
# suposem un m xim nombre de per odes (20)
l2<-c(l2.1,l2.2,l2.3)  # vl2 = vector l2.1,l2.2,...
## idem amb L1
if (f((B1/C1),A1,B1,C1) <= L1[1]){l1.1<-dates[1]}
if (f((B1/C1),A1,B1,C1) > L1[1]){
  l1.1<-floor(uniroot(function(x)
  {f(x,A=A1,B=B1,C=C1)-L1[1]},c(l2.1,2000),
  tol=1e-10)$root)
}
if (f((B1/C1),A1,B1,C1) <= L1[2]){l1.2<-dates[4]}
if (f((B1/C1),A1,B1,C1) > L1[2]){
  l1.2<-floor(uniroot(function(x)
  {f(x,A=A1,B=B1,C=C1)-L1[2]},c(l2.2,2000),
  tol=1e-10)$root)
}
if (f((B1/C1),A1,B1,C1) <= L1[3]){l1.3<-dates[7]}
if (f((B1/C1),A1,B1,C1) > L1[3]){
  l1.3<-floor(uniroot(function(x)
  {f(x,A=A1,B=B1,C=C1)-L1[3]},c(l2.3,2000),
  tol=1e-10)$root)
}
##l1.3<-floor(uniroot(function(x)
# suposem un m xim nombre de per odes (20)
l1<-c(l1.1,l1.2,l1.3)  # vector l1.1,l1.2,...
##list(l1=l1,l2=l2)
M<-matrix(c(l1,l2),nrow=1,ncol=6)
colnames(M)<-c("l1.1","l1.2","l1.3","l2.1",
"l2.2","l2.3")
M
}

###------ Taula de punts de tall, variant A

taula.li<-function(vA=(8:24),L1=c(15,18,18),
L2=c(25,25,25),
dates=c(100,200,300,100,200,300,100,200,300))
{
k<-length(vA)
taula<-matrix(rep(0,k*7))
dim(taula)<-c(k,7)
taula[1,]<- cbind(A=vA[1],arrels(A=vA[1],L1=L1,
L2=L2,dates=dates))
for(j in 1:k)
  taula[j,]<- cbind(A=vA[j],arrels(A=vA[j],L1=L1,
  L2=L2,dates=dates))
```

```
  colnames(taula)<-c("A","l1.1","l1.2","l1.3",
  "l2.1","l2.2","l2.3")
  taula<-data.frame(taula)
  taula
}


###------ Distribucions possibles de les vaques

#lleis: 1=unif,2=binom,3=trian.1,4=triang.2,5=parab


A <- 8:24            ## cal fixar-ho
h <- length(A)       ## ha de ser senar
## llei 1: X unif discreta entre 8 i 24
d1<-function(h){
        b<-1/h
        d1 <- rep(b,h)
        d1}

## llei 2: X+8, on X binomial(16,1/2)
d2 <- function(h){
        d2 <- dbinom(A-A[1],size=h-1,prob=0.5)
        d2}

## llei 3.1, triangular disminuint un 1%
d3.1<-function(h){
        z <- (h-1)/2
        p1 <- (100+(z+1)*z)/(1+2*z)
        d3.1 <- c( seq(p1-z,p1, by=1),
        seq(p1-1,p1-z,by=-1))/100
        d3.1}

## llei 3.2, triangular disminuint un 0.5%
d3.2<-function(h){
        z <- (h-1)/2
        p2 <- (100+(z+1)*z/2)/(1+2*z)
        d3.2 <- c(seq(p2-z/2,p2,by=0.5),
            seq(p2-0.5,p2-z/2,by=-0.5))/100
        d3.2}

d4 <- function(h){
        c <- 1
        m <- (A[h]+A[1])/2
        fA <- -c*(A-m)**2 +70
        d4 <- fA/sum(fA)
        d4}
```

```
lleis <- data.frame(A=A,d1=d1(),d2=d2(),
        d3.1=d3.1(),d3.2=d3.2(),d4=d4())

###------------------------- Simvaca

simvaca<-function(par.sim)
 {
  # recuperem els valors dels par metres
   aleat=par.sim[1]; llei=par.sim[2:18]; A0=par.sim[19];
   a=par.sim[20];  p=par.sim[21]; int=par.sim[22];
   m.gest=par.sim[23];  s.gest=par.sim[24];
   TM0=par.sim[25];  sec=par.sim[26];
  #
   if (aleat==1) A1<-sample(8:24,1,prob=llei,
    replace=TRUE)
   else  A1 <-A0
  #
  B1<-0.208; C1<-0.002
  A2<-A1*1.54; B2<-0.179; C2<-0.003
  A3<-A1*1.47; B3<-0.209; C3<-0.0036
 # seq ncia de 20 A1,...,A20, idem B, idem C
  As<-c(A1,A2,rep(A3,18))
  Bs<-c(B1,B2,rep(B3,18))
  Cs<-c(C1,C2,rep(C3,18))
 ### c lcul dels potencials
   t<-1:305
   pot1<-sum(f(t,A1,B1,C1))
   pot2<-sum(f(t,A2,B2,C2))
   pot3<-sum(f(t,A3,B3,C3))
### generem la resta d'objectes aleatoris
# Tmort amb llei Weibull sense par metres
# variables
# scale (scl) tal que la mediana s 1000,
# fixat shape =2
   scl<- 1000*(log(2))^(-1/2)
   Tmort<-floor(rweibull(1,shape=2,scale=scl))
# inici de cicle reproductiu
# imposem "ic" (interval de ciclat) Weibull
# despla ada amb inici 40
# shape=a valor de par metre introdu t
# c lcul scale= b de "ic" imposant mediana=60
# (20 a l'origen)
   b<- 20*(log(2))^(-1/a)
   ic<-c(0, 40 + floor(rweibull(19,shape=a,scale=b)))
# nombre d'inseminacions fallides (ni)
# genera 19 ni (incl s 0= a la primera va be) i indeps
```

```
   ni<-c(0,rgeom(19,p))
# genera 20 per odes de gestaci , indeps
   gest<-c(0,rnorm(19,m.gest,s.gest))
# genera el valor Bernoulli (0:no es ven, 1:S )
## nom s s'aplica amb mort no for ada
   val<-rbinom(1,1,TM0)
# iniciem els vectors de temps de la vaca a "0"
   Tc<-rep(0,20)
   Tig<-rep(0,20)
   Tpart<-rep(0,20)
   TL1<-rep(0,20)
   TL2<-rep(0,20)
   Tsec<-rep(0,20)
# calculem els temps posteriors
   for (j in 2:20)
     {
      Tc[j]          <- Tpart[j-1]+ic[j]
      Tig[j]         <- Tpart[j-1]+ ic[j]+ ni[j]*int
      # nom s secat biol gic
      Tsec[j-1]      <-  Tig[j]+sec
      Tpart[j]       <- ceiling(Tig[j]+gest[j])
     }
   Tsec[20]<-Tpart[20]+ 300
# resultats finals
   rs<-list(As=As,Bs=Bs,Cs=Cs,pot1=pot1, pot2=pot2,
            pot3=pot3,Tc=Tc,ni=ni,Tig=Tig,Tpart=Tpart,
            Tsec=Tsec,TMORT=Tmort,val=val )
   rs    # format llista
  }

#### Output of one simulated cow ('rs')

###------------------------- fbp

####  protocol tr bol

fbp<-function(rs,li,par.prtcl,par.preus)
 {
  # recuperem els valors dels par metres de protocol
   L1=par.prtcl[1:3];  L2=par.prtcl[4:6];
   dates=par.prtcl[7:15]; alpot=par.prtcl[16:24];
  # recuperem els valors dels par metres de preus
   cost1=par.preus[1:2]; cost0=par.preus[3];
   pll=par.preus[4]; pc=par.preus[5]; pv=par.preus[6];
   esp=par.preus[7];
   graf=par.preus[8];
   As<-rs$As
```

```
   Bs<-rs$Bs
   Cs<-rs$Cs
   pot1<-rs$pot1
   pot2<-rs$pot2
   pot3<-rs$pot3
   Tc<-rs$Tc
   Tig<-rs$Tig
   Tpart<-rs$Tpart
   Tsec<-rs$Tsec
   val<-rs$val
   TMORT<-rs$TMORT
 # alarmes per potencial baix, als 20 per odes
  al1<-c(pot1<alpot[1],pot2<alpot[4],rep(pot3<alpot[7],18))
    # alarmes a la data 1 (100)
  al2<-c(pot1<alpot[2],pot2<alpot[5],rep(pot3<alpot[8],18))
    # alarmes a la data 2 (200)
  al3<-c(pot1<alpot[3],pot2<alpot[6],rep(pot3<alpot[9],18))
    # alarmes a la data 3 (300)
#  temps en que s'arriba als llindars de baixa produccio
#  s'han calculat amb la funcio arrels i s'han tabulat a la
#  taula " li "
#  lj.i punt que determina el llindar de baixa
#  produccio sota Lj (j=1,2)  al per ode i  (j=1,2)
#  determinem la fila i columna de la taula.li on cal anar
#  a buscar
   A1<-As[1]
   fila<-A1-7
   l2.1<-li[fila,5]
   l2.2<-li[fila,6]
   l2.3<-li[fila,7]
   l1.1<-li[fila,2]
   l1.2<-li[fila,3]
   l1.3<-li[fila,4]
# suposem un maxim nombre de periodes (20)
   vl2<-c(l2.1,l2.2,rep(l2.3,18)) #vl2=vector l2.1,l2.2,...
   vl1<-c(l1.1,l1.2,rep(l1.3,18)) #vl1=vector l1.1,l1.2,...

   dat1<-c(dates[1],dates[4],rep(dates[7],18))
   dat2<-c(dates[2],dates[5],rep(dates[8],18))
   dat3<-c(dates[3],dates[6],rep(dates[9],18))

### calcul dels temps de sacrifici possibles, sequenciats
# calcul dels vectors TL1 i TL2
   TL1<-Tpart+vl1
   TL2<-Tpart+vl2

   log01 <-(Tig[-1]-TL1[-20])>0
```

```
## inici de gestacio posterior al llindar L1=15 per defecte
   log02 <-(Tig[-1]-TL2[-20])>0
## inici de gestacio posterior al llindar L2=25 per defecte
   log1<-(TL2[-20]-Tpart[-20])<=dat2[-20]) & (al1[-20]==1)
   log2<-(TL2[-20]-Tpart[-20]) > dat2[-20]  &
         (TL2[-20]-Tpart[-20]) <= dat3[-20] & al2[-20] == 1
   log3<-(TL2[-20]-Tpart[-20]) > dat3[-20]  & al3[-20] == 1

   log<-log01+(1-log01)*log02*(log1+log2+log3)
## vector l gic amb els per odes dolents, de sacfrifici
   log<-c(log,1)
   w1<-TL1*log
   w1<-w1[w1!=0]
   Tsacr<-round(w1[1])
 # ja te en compte el sacrifici a TL[20], si
 # la FI no es anterior
  bf<-0       ## benefici=0
  y<-0        ## y contindra la produccio de llet
  t<-0        ## comptador de temps d'una vaca concreta
 #-------------------------# calcul de TFI

  TFI<-min(TMORT,Tsacr)
 #------------------ # sequenciacio dels temps de la vaca
  taux  <- 0:(TFI) ;          #length(taux)
  ITc   <-rep(0,TFI+1) ;      #length(ITc)
  ITL1  <-rep(0,TFI+1) ;      #length(ITL1)
  ITL2  <-rep(0,TFI+1) ;      #length(ITL2)
  ITsec <-rep(0,TFI+1) ;      #length(ITsec)
  ITpart<-rep(0,TFI+1) ;      #length(ITpart)
 #
  ITFI<-c(rep(0,TFI-1),1)     # indicadors de TFI
  for(i in 2:20) ITc<-ITc+(taux==round(Tc[i]))
  for(i in 1:20) ITL1<-ITL1+(taux==round(TL1[i]))
  for(i in 1:20) ITL2<-ITL2+(taux==round(TL2[i]))
  if ( TFI ==Tsacr)
    {
       k <- sum(ITL1)
       esp <- 0
       Tsec[k]<-TFI
       for(i in 1:k) ITsec<-ITsec+(taux==round(Tsec[i]))
       for(i in 1:k) ITpart<-ITpart+(taux==round(Tpart[i]))
    }
  else {
  if ( TFI == TMORT )
    {
       for(i in 1:20) ITsec<-ITsec+(taux==Tsec[i])
       for(i in 1:20) ITpart<-ITpart+(taux==Tpart[i])
```

```
       k<-sum(ITpart)
    # esp  s  el temps per defecte, en aquest cas
    }  }


 seq<-cumsum(ITpart)-cumsum(ITsec)
#seq te valor 1 als trams productius i zero als trams seca
   per<-cumsum(ITpart)
 # "per" pren valors "11111...22222 ..." indicant el cicle
 # un cicle pot incloure el periode productiu i el
 # periode seca
 #--------------------  # c lcul de la produccio
   dies.per<-as.data.frame(table(per))
 # "dies.per"  s  el nombre de dies de cada periode
   z<-NULL
   for(j in 1:k)
     {z<-c(z,1:dies.per[j,2])}
      # x torna a comen ar a 1, a cada periode
 t<-0:(length(seq)-1+esp)  #length(t) inclou temps d'espera
 x0<-z*(seq!=0)           #length(x0)# es 0 quan "seq" es 0
 # produccio de llet
 y0<-f(x0,A=As[per],B=Bs[per],C=Cs[per])  #length(y0)
 x<-c(x0,rep(0,esp))       # afegeixo esp       # length(x)
 y<-c(y0,rep(0,esp))       # afegeixo esp       # length(y)
 #---------------------      # calcul de benefici # bf
 prv<- sum(y)             # produccio total 1 vaca en litres
 tseca<-sum(y0==0)  # temps seca, sense "esp" ni primer dia
 tprod<- TFI - tseca ## tprod + tseca = TFI  # no entra esp
 cost.pr<- sum(y!=0)*cost1[1] + cost1[2]*sum(y)
 if (TFI==Tsacr){bf<- bf-pc+pv+prv*pll-cost0*tseca-cost.pr}
 if (TFI==TMORT){
      bf<- bf-pc+pv*val+prv*pll-cost0*tseca-cost.pr}
 # recordem que "val" utilitza TM0=50% per defecte
 #
   bfvd<-bf/TFI             # benefici diari vaca
   bfvd.c<-bf/length(t)     # benefici diari corregit
   prvd<-prv/TFI            # produccio diaria
   prvd.c<-prv/length(t)    # prod. diaria corregida
    res<-list(tv=t,y=y,Tc=rs$Tc,Tig=rs$Tig,TFI=TFI,
              Tsacr=Tsacr,np=k,prv=prv,prvd=prvd,
              prvd.c=prvd.c,bfv=bf,bfvd=bfvd,bfvd.c=bfvd.c)
    res
 }

availLleis<-c(Unif="d1", Binomial="d2", Triangular1="d3.1",
            Triangular2="d3.2", Parabolica="d4")
#availLleis<-c(Unif="d1")
```

```
############################
######### 2a part #########
############################

### An lisi de la dependencia

### Dependencia de L1  ('dep.fL1.1')   (15)

#  Aquest codi s'ha d'executar cada vegada que volguem
#  mirar la depend ncia de L1,
# canviant algun dels molts parametres (excepte el vector
# vL1.1) es a dir, es pot analitzar la dependencia canviant
# Nsim, alpot, dates, costos, etc...

dep.fL1<-function(vL1.1,
       Nsim,
       alpot1.1,alpot2.1,alpot3.1,
       alpot1.2,alpot2.2,alpot3.2,
       alpot1.3,alpot2.3,alpot3.3,
       dates,
       cost1.1,cost1.2,cost0,pll,pc,pv,esp,graf,
       aleat,llei,A0,a,p,int,m.gest,s.gest,TM0,sec) {
 par.sim<-c(aleat,llei,A0,a,p,int,m.gest,s.gest,TM0,sec)
 cost1<-c(cost1.1,cost1.2)
 par.preus<-c(cost1,cost0, pll, pc, pv, esp, graf)
 alpot<-c(alpot1.1,alpot2.1,alpot3.1,alpot1.2,alpot2.2,
          alpot3.2, alpot1.3,alpot2.3,alpot3.3)

 mTFI <-rep(0,length(vL1.1))   #mTFI<-0
 mBFV  <-rep(0,length(vL1.1))  #mBFV<-0
 T <-rep(0,length(vL1.1))  #T<-0
  Y <-rep(0,length(vL1.1))  #Y<-0
  BF<-rep(0,length(vL1.1))
  id.vaca<-0
  while(id.vaca <= Nsim-1)
   {
    rs<-simvaca(par.sim)
  # simulo 1 vaca i, fixada, calculo els "bf" amb diversos
  # valors de "L1.1"
    for (i in (1:length(vL1.1)))
     {
      #rs<-simvaca()
      # si ho poso aqu , genera mostra diferent per cada
      # valor de L1.1

      ## -------------------------   i=1
```

```
      L1.1=vL1.1[i]
      L1.2= L1.1+3;  L1.3= L1.2; # per exemple 15,18,18
      L2.1= L1.1+10;  L2.2= L2.1;  L2.3= L2.1;
      # per exemple 25,25,25
      L1<-c(L1.1,L1.2,L1.3)
      L2<-c(L2.1,L2.2,L2.3)
      par.prtcl<-c(L1,L2,dates,alpot)  ## longitud 24
      li.act<-taula.li(vA=(8:24),L1=par.prtcl[1:3],
              L2=par.prtcl[4:6],dates=par.prtcl[7:15])
      ## ----------------------
      parcial<-fbp(li=li.act,rs,par.prtcl,par.preus)
      T[i]<-  T[i]+parcial$TFI
      Y[i]<-  Y[i]+sum(parcial$y)
      BF[i]<- BF[i]+parcial$bfv
     }
    id.vaca<-id.vaca+1
   }

  meanBFV<-  BF/Nsim
  meanTFI<-  T/Nsim
  BFd<-BF/T
  PRd<-Y/T
  arxiu<-NULL
  for (i in (1:length(vL1.1)))
   {
    arxiu<-rbind(arxiu, c(sec=sec, L1.1=vL1.1[i],
        cost1.1=cost1[1] ,cost1.2=cost1[2] ,
        cost0=cost0,pll=pll,pc=pc,pv=pv,Nsim=Nsim,
        prvd=PRd[i],bfvd=BFd[i],
        mBFV=meanBFV[i] ,mTFI=meanTFI[i]))
   }
  arxiu
}


part2<-function(h,...){

results<-data.frame(dep.fL1(vL1.1=13:20,Nsim=svalue(wNsim),
            alpot1.1=6000,alpot2.1=7000,alpot3.1=8000,
            alpot1.2=8500,alpot2.2=9000,alpot3.2=11000,
            alpot1.3=8500,alpot2.3=9000,alpot3.3=11000,
            dates=c(100,200,300,100,200,300,100,200,300),
        cost1.1=svalue(wcost1.1),cost1.2=svalue(wcost1.2),
        cost0=svalue(wcost0),pll=svalue(wpreullet),
        pc=svalue(wpreucompra),pv=svalue(wpreuventa),
        esp= 20,graf=0,aleat=1,llei=
        do.call(availLleis[svalue(wllei)],list(17)),
```

```
        A0=16,a=2,p=svalue(wp),int=svalue(wint),m.gest=280,
        s.gest=3,TM0=0.5,sec=220 ))

plot(results$L1.1,results$bfvd,
        xlab="low production threeshold",
        ylab="Daily profit",
        type="b", col="blue",
        ylim=c(min(results$bfvd),max(results$bfvd))
        )
     title(main="Profit over low production threshold")
}


####################################
######### CREACIO WIDGETS #########
####################################

library(gWidgets)
library(gWidgetsRGtk2)
library(cairoDevice) # nou paquet
library(lattice)

wmgest<-glabel(text="280", editable=FALSE, handler=NULL)
wsgest<-glabel(text="3", editable=FALSE, handler=NULL)
wtm<-glabel(text="0.5", editable=FALSE, handler=NULL)
wsec<-glabel(text="220", editable=FALSE, handler=NULL)
wllei<-gradio(names(availLleis), horizontal=FALSE,
    handler=part2)
wp<-gspinbutton(from=0,to=1,by=0.05, value=0.15,
    handler=part2)
wint<-gspinbutton(from=0,to=50,by=1,value=21,handler=part2)

wll15.1.1<-glabel(text="15", editable=FALSE, handler=NULL)
wll15.1.2<- glabel(text="18", editable=FALSE, handler=NULL)
wll15.1.3<- glabel(text="18", editable=FALSE, handler=NULL)
wll15.2.1<- glabel(text="25", editable=FALSE, handler=NULL)
wll15.2.2<- glabel(text="25", editable=FALSE, handler=NULL)
wll15.2.3<- glabel(text="25", editable=FALSE, handler=NULL)

wll16.1.1<-glabel(text="16", editable=FALSE, handler=NULL)
wll16.1.2<- glabel(text="19", editable=FALSE, handler=NULL)
wll16.1.3<- glabel(text="19", editable=FALSE, handler=NULL)
wll16.2.1<- glabel(text="26", editable=FALSE, handler=NULL)
wll16.2.2<- glabel(text="26", editable=FALSE, handler=NULL)
wll16.2.3<- glabel(text="26", editable=FALSE, handler=NULL)

wll17.1.1<-glabel(text="17", editable=FALSE, handler=NULL)
```

```
wll17.1.2<- glabel(text="20", editable=FALSE, handler=NULL)
wll17.1.3<- glabel(text="20", editable=FALSE, handler=NULL)
wll17.2.1<- glabel(text="27", editable=FALSE, handler=NULL)
wll17.2.2<- glabel(text="27", editable=FALSE, handler=NULL)
wll17.2.3<- glabel(text="27", editable=FALSE, handler=NULL)

wdat1.1<- glabel(text="100", editable=FALSE, handler=NULL)
wdat2.1<- glabel(text="100", editable=FALSE, handler=NULL)
wdat3.1<- glabel(text="100", editable=FALSE, handler=NULL)
wdat1.2<- glabel(text="200", editable=FALSE, handler=NULL)
wdat2.2<- glabel(text="200", editable=FALSE, handler=NULL)
wdat3.2<- glabel(text="200", editable=FALSE, handler=NULL)
wdat1.3<- glabel(text="300", editable=FALSE, handler=NULL)
wdat2.3<- glabel(text="300", editable=FALSE, handler=NULL)
wdat3.3<- glabel(text="300", editable=FALSE, handler=NULL)

walpot1.1<-glabel(text="6000",editable=FALSE,handler=NULL)
walpot2.1<-glabel(text="7000",editable=FALSE,handler=NULL)
walpot3.1<-glabel(text="8000",editable=FALSE,handler=NULL)
walpot1.2<-glabel(text="8500",editable=FALSE,handler=NULL)
walpot2.2<-glabel(text="9000",editable=FALSE,handler=NULL)
walpot3.2<-glabel(text="11000",editable=FALSE,handler=NULL)
walpot1.3<-glabel(text="8500",editable=FALSE,handler=NULL)
walpot2.3<-glabel(text="9000",editable=FALSE,handler=NULL)
walpot3.3<-glabel(text="11000",editable=FALSE,handler=NULL)

wcost1.1<- gspinbutton(from=1,to=6,by=0.5, value=2.5,
            handler=part2)
wcost1.2<- gspinbutton(from=0.05,to=1,by=0.05, value=0.15,
            handler=part2)
wcost0<- gspinbutton(from=1,to=6,by=0.5, value=3,
          handler=part2)
wesp<- glabel(text="20", editable=FALSE, handler=NULL)
wpreucompra <- gcombobox(c(1600,1700,1800,1900,2000),
          selected = 3, editable = FALSE, coerce.with=
          as.numeric, handler = part2, action = NULL)
wpreullet <- gcombobox(c(0.10,0.15,0.20,0.25,0.30,0.35,0.40,
          0.45, 0.50,0.55,0.60), selected = 5, editable =
          FALSE, coerce.with=as.numeric, handler = part2,
          action = NULL)
wpreuventa <- gcombobox(c(500,600,700,800,900,1000),
          selected = 4, editable = FALSE, coerce.with=
          as.numeric, handler = part2, action = NULL)

wNsim<-gspinbutton(from=5,to=100000,by=1, value=10,
        handler=part2)
refresh<-gbutton(text="REFRESH",border=TRUE,handler=part2)
```

```
window<-gwindow("FINAL GUI") #aqui ja executa la finestra

BigGroup <- ggroup(cont=window,expand=FALSE)
group1 <-ggroup(horizontal=FALSE, use.scrollwindow = FALSE,
          container=BigGroup)
tmp <- gframe("PAR.SIM", container=group1)
tmp <-gframe("Average of gestation time:",container=group1)
add(tmp,wmgest)
tmp <- gframe("Deviation of gestation time:",
        container=group1)
add(tmp,wsgest)
tmp <- gframe("Prob. of usable meat:", container=group1)
add(tmp,wtm)
tmp <- gframe("Drying pregnancy:", container=group1)
add(tmp,wsec)
tmp <- gframe("Law that defines the arrival of cows to the
        farm:", container=group1)
add(tmp,wllei)
tmp<-gframe("Insemination probability:", container=group1)
add(tmp,wp)
tmp<-gframe("Days between inseminations:",container=group1)
add(tmp,wint)

group2 <- ggroup(horizontal=FALSE,use.scrollwindow=FALSE,
        container=BigGroup)

tmp <- gframe("PAR.COST", container=group2)
tmp <- gframe("Milk price:", container=group2)
add(tmp, wpreullet)
tmp <- gframe("Price of a new cow:", container=group2)
add(tmp, wpreucompra)
tmp <- gframe("Price of selling a cow:", container=group2)
add(tmp, wpreuventa)
tmp <- gframe("Cost0 (dried cow):", container=group2)
add(tmp, wcost0)
tmp <- gframe("Cost1 (no dried cow):",
        container=group2)
add(tmp, wcost1.1)
add(tmp, wcost1.2)
tmp<-gframe("Waiting time of a new cow:",container=group2)
add(tmp, wesp)

group3<-ggroup(horizontal=FALSE, use.scrollwindow = FALSE,
           container=BigGroup)
tmp <- gframe("PAR.PRTCL", container=group3)
group3.1<-gexpandgroup(text="Thresholds depending on vL1.1",
```

```
        markup = FALSE, horizontal=TRUE, handler = NULL,
        action = NULL,container = group3)
tmp <- gframe("si vL1.1=15", container=group3.1)
tmp <- gframe("Threshold 1", container=group3.1)
add(tmp,wll15.1.1)
add(tmp,wll15.1.2)
add(tmp,wll15.1.3)
tmp <- gframe("Threshold 2", container=group3.1)
add(tmp,wll15.2.1)
add(tmp,wll15.2.2)
add(tmp,wll15.2.3)
tmp <- gframe("si vL1.1=16", container=group3.1)
tmp <- gframe("Threshold 1", container=group3.1)
add(tmp,wll16.1.1)
add(tmp,wll16.1.2)
add(tmp,wll16.1.3)
tmp <- gframe("Threshold 2", container=group3.1)
add(tmp,wll16.2.1)
add(tmp,wll16.2.2)
add(tmp,wll16.2.3)
tmp <- gframe("si vL1.1=17", container=group3.1)
tmp <- gframe("Threshold 1", container=group3.1)
add(tmp,wll17.1.1)
add(tmp,wll17.1.2)
add(tmp,wll17.1.3)
tmp <- gframe("Threshold 2", container=group3.1)
add(tmp,wll17.2.1)
add(tmp,wll17.2.2)
add(tmp,wll17.2.3)

tmp <- gframe("Period dates 1", container=group3)
add(tmp,wdat1.1)
add(tmp,wdat1.2)
add(tmp,wdat1.3)
tmp <- gframe("Period dates 2", container=group3)
add(tmp,wdat2.1)
add(tmp,wdat2.2)
add(tmp,wdat2.3)
tmp <- gframe("Period dates 3", container=group3)
add(tmp,wdat3.1)
add(tmp,wdat3.2)
add(tmp,wdat3.3)
tmp <- gframe("Potential alarms of period 1",
        container=group3)
add(tmp,walpot1.1)
add(tmp,walpot1.2)
add(tmp,walpot1.3)
```

```
tmp <- gframe("Potential alarms of period 2",
        container=group3)
add(tmp,walpot2.1)
add(tmp,walpot2.2)
add(tmp,walpot2.3)
tmp <- gframe("Potential alarms of period 3",
        container=group3)
add(tmp,walpot3.1)
add(tmp,walpot3.2)
add(tmp,walpot3.3)

tmp <- gframe("Number of simulations:", container=group3)
add(tmp,wNsim)
tmp <- gframe("New sample", container=group3)
add(tmp, refresh)
add(BigGroup, ggraphics())
```

Appendix D

Reproductive cycles of a cow, model II: simulations and profit analysis

By Aureli Alabert, Sergio Calsamiglia, Lorena Castillejos, and Mercè Farré.

We assume that the milk production over time, in each cycle, is well adjusted by the *Wood model*

$$f(x) = A\, x^B\, e^{-C\, x},$$

where parameters A, B and C depend on the cycle, with the restrictions given in Table D.1.

Table D.1: Coefficients in the Wood model.

	1st	2nd	others
A	$A_1 \in \{10, ..., 24\}$	$A_2 = 1.54\, A_1$	$A_3 = 1.47\, A_1$
B	$B_1 = 0.208$	$B_2 = 0.179$	$B_3 = 0.209$
C	$C_1 = 0.002$	$C_2 = 0.003$	$C_3 = 0.0036$

D.1 Variables: hypotheses and parameters

In item 1, we describe the basic random variables defining the model. In item 2, we introduce more parameters that influence the farmer decisions. To analyze the whole behavior of a cow's evolution, more variables are needed, but all that new variables can be obtained as functions of the variables in item 1 and the parameters in item 2, we specify these new variables in item 3.

1. Basic random variables and its parametrization.

   (a) We assume that parameter A_1 can be chosen at random in its rang of possible values, that is, we assume that follows a discrete uniform distribution in $[10, 24]$. Nevertheless, we allow that this range could be much more restricted if it's needed.

      i. Once the random value is obtained, if the resulting random values are 10 or 11, we decide that this cow presents a *low production pattern* and must be sacrificed at the end of period 1. With this rule, around a 10% of animals will be sacrificed by this reason.

      ii. If $aleat = 1$, the values of parameter A_1 are chosen randomly, as described above, but, if $aleat = 0$, then A_1 is a fixed value denoted by A_0, being 17 its default value.

   (b) Initiation of the reproductive cycle ic

   - *Hypothesis:* In each period, ic follows a Weibull distribution (with origin translated to 40 (days)) with *shape* parameter $a = 2$ (as default value) and *scale* parameter $b = 20(\log 2)^{-\frac{1}{a}}$. This condition provides that a 50% of the cows initiate before 60 days. All them are independent and identically distributed.
   - *Parameters:* a, with 2 as a default value.

   (c) Number of unsuccessful inseminations ni

   - *Hypothesis:* In each period, n_i follows a geometric distribution, taking values 0,1,2,..., with parameter p. All them are independent and identically distributed.
   - *Parameter:* p, with 0.15 as default value.

   (d) Gestation time $gest$

   - *Hypothesis:* In each period, $gest$ follows a Gaussian distribution with parameters $m.gest$ and $s.gest$. All them are independent and identically distributed.
   - *Parameters:* $m.gest$ and $s.gest$, with 280 days and 3 days, respectively, as default values.

   (e) Non-forced death time $Tmort$

   - *Hypothesis:* It follows a Weibull distribution with $shape = 2$ and $scale = 1000(\log 2)^{-\frac{1}{a}}$ providing that 10% ($\approx$) will die in the first year and a 50% ($\approx$) in 1000 days.

   Random variables in different items are all independent too.

2. "Deterministic" parameters. Their values can be modified by farmers decisions, economic reasons and others,but not in a random way.

- int : The interval between consecutive "inseminacions", with 21 days as the default value.
- sec : The interval after initiating pregnancy for the "biological drying", with 220 days as the default value.
- L : The threshold of low production for the "economical drying", with 15 liters as the default value.
- $sacr$: The threshold for the "economical sacrifice" (number of days after labor if there's no pregnancy), with 200 days as the default value.
- $cost0$: The constant daily cost of a cow in its dry period, with 3 euros as the default value.
- $cost1 = (cost1_1, cost1_2)$: In the productive period, we assume a fixed cost ($cost1_1$) (being 2.5 euros the default value) and a variable cost with multiplicative factor $cost1_2$ times the milk produced (0.15 is that factor's default value.)
- pll : The price of the liter of milk paid to the farmer, with 0.32 euros as the default value.
- pc : The purchase price of the cow paid by the farmer, with 1800 euros as the default value.
- pv : The selling price of the cow paid to the farmer, with 800 euros as the default value.
- $TM0$: In the *non-forced death*, this tax is the probability of having a disease that prevents the meat sold, with 0.5 (50%) as the default value.
- esp : The expecting time for having a new cow. It is assumed to be 0 when the sacrifice is programmed and 20, as default value, when there's an unexpected or *non-forced death*.

3. New random variables, defined as functions of the variables in item 1 and the parameters in item 2.

   - From the Wood model function and the parameter values, taking into account the random condition on coefficient A_1 and the the threshold value of L, we compute using numerical methods the points where the "economical drying" is prescribed, for each period. The threshold $l.j$, for the period j, is defined as an integer approximation of the greatest of the two roots of the following function
   $$g_j(x) = f(x, A_j, B_j, C_j) - L\,,$$
   that is,
   $$l.j = \lfloor x \rfloor \quad \text{where} \quad g_j(x) = 0.$$

- From the tax $TM0$, we define a random binary variable, *val*, taking the value 1 with probability $TM0$. This variable only applies when the death is due to *non forced* causes. If the *val* value is 0, then the cow cannot be sold.
- By applying functional dependencies, we deduce new random variables
$$Tpart_j,\ Tc_j,\ Tig_j,\ Tsec_j,\ TL_j,$$
in a recursive way.
The time origin is the first labor time: $Tpart_1 = 0$. For coherence, all the time values corresponding to events preceding the first labor time are fixed to be zero: $Tc_1 = 0$ and $Tig_1 = 0$. To start the recursive process, assume that we have all the values until $Tpart_j$, so the period j starts and we obtain:
  - The time for *economic drying* in this period is
$$TL_j = Tpart_j + l.j$$
  - The starting point of the new $j+1$-reproductive cycle is
$$Tc_{j+1} = Tpart_j + ic_{j+1}$$
  - The starting point of the new $j+1$ pregnancy is
$$Tig_{j+1} = Tpart_j + ic_{j+1} + ni_{j+1} \times int$$
  - The time for *biological drying* is
$$Tsec_j = \min\{TL_j, Tig_j + sec\}$$
  - The cycle is completed with the next labor time $j+1$
$$Tpart_{j+1} = Tig_{j+1} + gest_{j+1}$$

  Notice that there is one ordered sequence
$$Tpart_j \leq Tc_{j+1} \leq Tig_{j+1} \leq Tig_{j+1} + sec \leq_{(*)} Tc_{j+1}$$

Remarks:
The equality $(*)$ is not sure, but has a high probability ...(–to discuss–)

D.2 Individual simulations

Model of a cow implemented with $\mathcal{R}$. Schematic description.

D.2.1 The times

The following R commands calls for function `simvaca()` with the following default parameter values ($a = 2, p = 0.15, int = 21, sec = 220, L = 15, m.gest = 280, s.gest = 3, aleat = 1, A0 = 17, TM0 = 0.5$).

```
source("C:/Users/farre/Desktop/vaques/v2/simvaca.R")
rs<-simvaca();rs
```

This code loads the function `simvaca()` and executes it, giving the output object `"rs"` decribing the simulated cow:

$$Tc,\ ni,\ Tig,\ Tpart,\ Tsec,\ TL,\ Tmort,\ val,\ A,\ B,\ C$$

Here, no restrictions of sacrifice are imposed and 20 periods are described ...

As an example, we can see one example tabulated in Table **??**.

Table D.2: Periods for a cow, the *non-forced death* would arribe in 1119 days, provided that no sacrifice is prescribed before

per	Tc	ni	Tig	Tpart	Tsec	TL	Tmort	Tmbp	val	As	Bs	Cs
1	0	0	0	0	329	751	1119	9597	1	17.00	.208	.0020
2	67	2	109	388	7 98	951	1119	9597	1	26.18	.179	.0030
3	473	5	578	863	1170	1365	1119	9597	1	24.99	.209	.0036
4	908	2	950	1229	1540	1731	1119	9597	1	24.99	.209	.0036
5	1299	1	1320	1600	1922	2102	1119	9597	1	24.99	.209	.0036
6	1681	1	1702	1986	2476	2488	1119	9597	1	24.99	.209	.0036
7	2046	10	2256	2536	2858	3038	1119	9597	1	24.99	.209	.0036
8	2596	2	2638	2916	3394	3418	1119	9597	1	24.99	.209	.0036
9	2964	10	3174	3451	3939	3953	1119	9597	1	24.99	.209	.0036
10	3509	10	3719	4004	4340	4506	1119	9597	1	24.99	.209	.0036
11	4057	3	4120	4397	4818	4899	1119	9597	1	24.99	.209	.0036
12	4451	7	4598	4879	5381	5381	1119	9597	1	24.99	.209	.0036
13	4949	13	5222	5501	6003	6003	1119	9597	1	24.99	.209	.0036
14	5552	16	5888	6168	6445	6670	1119	9597	1	24.99	.209	.0036
15	6225	0	6225	6511	7013	7013	1119	9597	1	24.99	.209	.0036
16	6564	19	6963	7243	7557	7745	1119	9597	1	24.99	.209	.0036
17	7316	1	7337	7618	8060	8120	1119	9597	1	24.99	.209	.0036
18	7672	8	7840	8119	8545	8621	1119	9597	1	24.99	.209	.0036
19	8178	7	8325	8606	9036	9108	1119	9597	1	24.99	.209	.0036
20	8669	7	8816	9095	9597	9597	1119	9597	1	24.99	.209	.0036

These output values will be used in the function that computes de *profit*.

D.2.2 Sacrifice and benefit for one cow

In the section above, we looked at the evolution of the cow, but no economical decision is taken.

The funcion `fbp` takes into account the economical inputs. The function decides, in each period j, if the cow must be sacrificed, that is, if

$$Tig_{j+1} - Tpart_j > sacr\,.$$

If this condition is satisfied, then the *economical sacrifice* is prescribed at the end of this period. Recall that another *sacrifice by low production* could be programmed when the production is too much low. We denote this time by $Tmbp$, and $Tmbp = TL_1$, when $A_1 = 10$ or 11, otherwise there is no restriction for this reason and $Tmbp$ is the maximum of the considered time interval. Therefore, the cow's end should be computed as

$$TFI = min\{Tmort, Tsacr, Tmbp\}\,.$$

Once TFI is determined and esp is added (if needed), then the daily production vector y is computed from 0 to TFI. The production prv and the economic profit bfv are given by: formulas:

$$\begin{aligned} prv &= \sum y && \text{(D.1)}\\ tseca &= \left[\sum 1_{\{y=0\}}\right] - esp && \text{(D.2)}\\ tprod &= TFI - tseca = \sum 1_{\{y\neq 0\}} && \text{(D.3)}\\ bfv &= prv \cdot pll - cost0 \cdot tseca - cost.pr - pc + pv \cdot val, && \text{(D.4)} \end{aligned}$$

where

$$cost.pr = cost1_1 \cdot \sum 1_{\{y\neq 0\}} + cost1_2 \cdot \sum y.$$

Recall that val can be 0 when the death is due to some kind of patologies (with a tax of 50%).

The following R commands call for function `"fbp"` with default parameters $(rs, sacr = 200, cost1 = 7, cost0 = 3, pll = 0.32, pc = 1800, pv = 800, esp = 20, graf = 0)$, where the first object, rs, is the `simvaca()` output.

```
source("C:/Users/farre/Desktop/vaques/v2/fbp.R")
res<-fbp(rs=simvaca());res
```

This code loads the function `fbp` and executes it, giving the output object `res`, that describes the important times and the economical indicators for the cow,

tv, y, Tc, Tig, TFI, $Tsacr$, $Tmbp$, np, prv, $prvd$, $prvd.c$, bfv, $bfvd$, $bfvd.c$

where:

- tv is the complete timing sequence for the cow, that is, a vector of length $TFI + esp$ (esp can be zero) and TFI is the end time.
- y is the complete milk production sequence.

- Tc and Tig (copied from the `simvaca()` output and used in the graphics).
- $Tsacr$ and $Tmbp$ are the sacrifice times. If $TFI < \min(Tsacr, Tmpb)$, then the death is due to *non forced* reasons.
- np is the number of complete or incomplete periods from 0 to TFI.
- prv is the total cow's milk production.
- $prvd = prv/TFI$ is the daily profit for this cow.
- $prvd.c = prv/(TFI + esp)$ is the corrected daily profit for this cow, adding the expecting time (esp).
- bfv is the global profit for this cow.
- $bfvd = bfv/TFI$ is the daily profit for this cow.
- $prvd.c = bfv/(TFI + esp)$ is the corrected daily profit for this cow, adding the expecting time (esp).

D.2.3 Table and figure for one cow simulation

The following code uses the output `res` of the `fbp` functions and gives the relevant indicators in a table format. Moreover, function `graf` is called to obtain the figure D.1.

```
res.table<-data.frame(res[-(1:4)]); res.table
source("C:/Users/farre/Desktop/vaques/v2/graf.R")
graf(res)
```

Table D.3: Simulation values for randomly chosen cow. Notice that this cow lasts 3 periods until sacrifice

	TFI	Tsacr	Tmbp	np	prv	prvd	prvd.c	bfv	bfvd	bfvd.c
1	1119	2488	9597	3	36118	32.28	31.68	2278.2	2.036	1.998

Subsequent simulations can give rise to really different outputs; see figure D.2. These graphics can be obtained by means of the following code.

```
pdf("ato4vaques.pdf")
par(mfrow=c(2,2)); fig<-1
while(fig<5){res<-fbp(simvaca());graf(res);fig<-fig+1}
par(mfrow=c(1,1));dev.off()
```

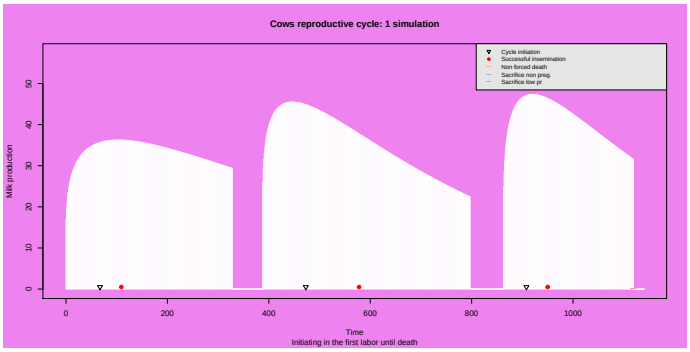

Figure D.1: Milk production corresponding to the cow to the previous table

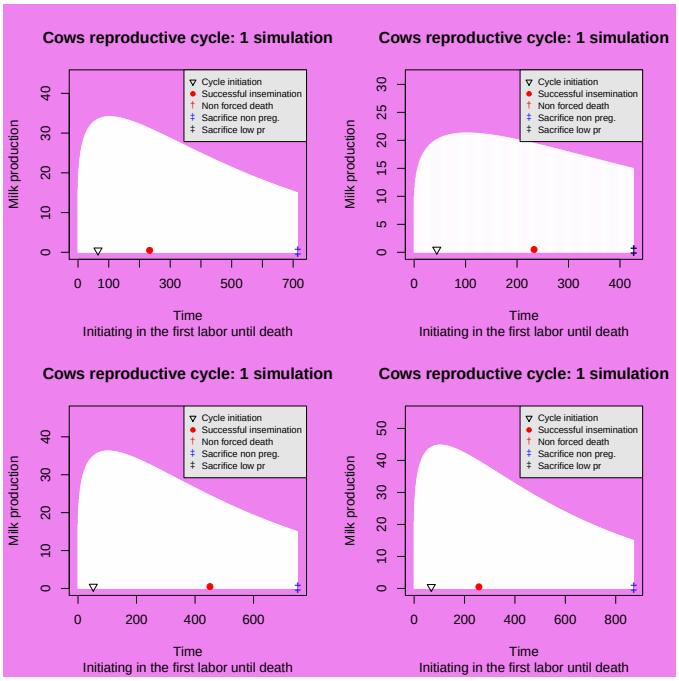

Figure D.2: Milk production corresponding to 4 different cows

D.3 Replicates

Next step consist in programming replicates for this experimental simulation process. Function `sims.consec()` fixes the parameters to replicate a selected number of simulations with the same parameters but sampling the random variables each time. Function `taula` tabulates as in table D.4.

```
source("C:/Users/farre/Desktop/vaques/v2/simsconsec.R")
sims1000<-sims.consec(Nsim=1000)  # gives 1000 replicates
taula1000<-taula(Nsim=1000)
```

The following code uses function `taula` listed above and summarizes some descriptives.

```
summary(taula(1000))
taula1000<-data.frame(taula1000)
mitjBFVD<-sum(taula1000$bfv)/sum(taula1000$TFI)
mitjBFVD
```

Table D.4: Ten replicates with the same parameter values

vaca	np	ni	TFI	prv	prvd	bfv	bfvd
1	3	1.67	1208	27353.33	22.64	-226.93	-0.19
2	4	3.25	1812	55992.27	30.90	4941.53	2.73
3	4	1.00	1758	68660.83	39.06	9433.47	5.37
4	4	1.00	1608	44824.11	27.88	2819.72	1.75
5	8	1.75	3180	95669.53	30.08	9022.25	2.84
6	4	1.25	1680	58007.78	34.53	6526.49	3.88
7	2	0.00	983	34259.30	34.85	3361.98	3.42
8	1	0.00	270	5415.10	20.06	-1157.17	-4.29
9	10	2.70	4339	189979.24	43.78	31592.36	7.28
10	4	1.25	1646	51174.11	31.09	4621.72	2.81

The following code uses the function `sims.consec(100)` to obtain 10 replicates and represent a piece of time

```
sims<-sims.consec(10)
temps<-1000:24000    ## time interval choosen
y<-sims$y[1000:24000]
graf2sim<- par(bg= "violet")
plot(temps,y,type="h",col="white")
par(graf2sim)
```

An output is represented in figure D.3.

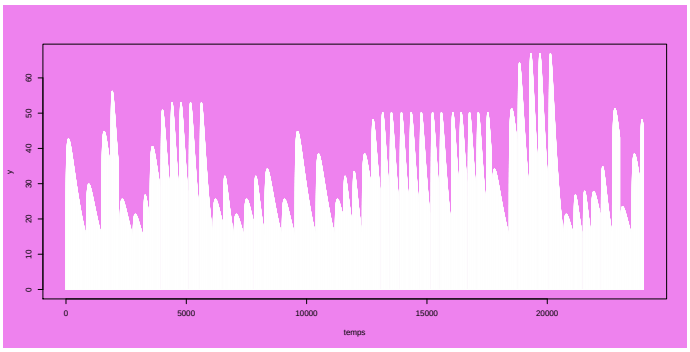

Figure D.3: Sequence of simulations

D.4 Analysis of parameter dependencies

Code to loads the `dep.func` and executes it to tabulate the results.

```
source("C:/Users/farre/Desktop/vaques/v2/depfunc.R")
vs<-seq(200,500,by=50)
a1000<-data.frame(dep.func(Nsim=1000,sacr=vs))
# writing the table in a excel spreadsheet
 require(xlsx)
 write.xlsx(a1000,"C:/Users/.../a1000.xlsx")
```

New results are stored (in an excel format, if needed).

```
a1000.2<-data.frame(dep.func(Nsim=1000,sacr=vs))
a1000.3<-data.frame(dep.func(Nsim=1000,sacr=vs))
a1000.4<-data.frame(dep.func(Nsim=1000,sacr=vs))
a1000.5<-data.frame(dep.func(Nsim=1000,sacr=vs))
```

Table D.5: Columns `mBFV` and `mTFI` store the mean profit and the mean life time of the same group of 1000 simulated cows farm, computed under different sacrifice times but keeping the other parameters constant.

sec	L	cost0 - cost$1_{1,2}$	pll	pc	pv	Nsim	sacr	prvd	bfvd	mBFV	mTFI
220	15	3 - 2.5 , 0.15	.32	1800	800	1000	200	31.6	1.26	987.2	785
220	15	3 - 2.5 , 0.15	.32	1800	800	1000	250	31.9	1.36	1122.5	827
220	15	3 - 2.5 , 0.15	.32	1800	800	1000	300	32.2	1.44	1240.9	864
220	15	3 - 2.5 , 0.15	.32	1800	800	1000	350	32.3	1.46	1304.6	892
220	15	3 - 2.5 , 0.15	.32	1800	800	1000	400	32.2	1.47	1341.5	915
220	15	3 - 2.5 , 0.15	.32	1800	800	1000	450	32.2	1.47	1366.5	927
220	15	3 - 2.5 , 0.15	.32	1800	800	1000	500	32.1	1.47	1374.9	938

Code for figure (final).

```
plot(a1000$sacr,a1000$bfvd,
     xlim=c(100,600),ylim=c(1.1,1.8),
     xlab="Unsuccessful pregnancy sacrifice threshold",
     ylab="Daily profit", type="b", col="blue")
     title(main="Profit over sacrifice threshold")
     points(a1000.2$sacr,a1000.2$bfvd,col="red")
     lines(a1000.2$sacr,a1000.2$bfvd,,col="red")
# and new lines added for a1000.3, etc.
```

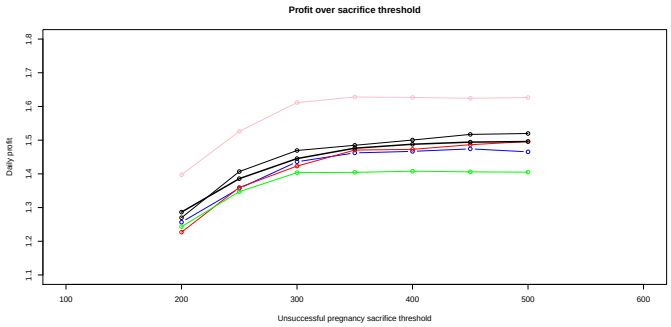

Figure D.4: Profit versus sacr. The thick line represents the results of 10000 simulations. Thin lines correspond to 1000 simulations each one.

Bibliography

[1] How to use the R-Tcl/Tk package, *The R-Tcl/Tk interface*, University of Copenhagen, Peter Dalgaart, 2001.

[2] RGtk2:, *A Graphical User Interface Toolkit for R*, Michael Lawrence (Fred Hutchinson Cancer Research Center) and Duncan Temple Lang (University of California), 2010.

[3] R bindings for Gtk 2.8.0 and above, *Package "RGtk2"*, Michael Lawrence and Duncan Temple Lang, 2013.

[4] gWidgets, *A Toolkit-Independent API for Building GUIs in R*, John Verzani, 2007.

[5] gWidgets, *Examples*, John Verzani, 2012.

[6] Function GUI, *Package 'fgui'*, Thomas Hoffmann, 2012.

[7] Introduction to Tcl, *R: Creating Graphical User Interfaces using the tcltk*, University of Waterloo, Adrian Waddell, 2010.

[8] Tcl/Tk tutorial, *www6.uniovi.es/tcl/tutorial/*, University of Oviedo, 1998.

[9] R Tcl/Tk Examples, *bioinf.wehi.edu.au/ wettenhall/RTclTkExamples/*, James Wettenhall, 2004.

[10] How to use the Qt package, *An Introduction to qtbase*, Michael Lawrence, 2012.

[11] Introduction to Qt, *Interfaces to the Qt framework from R*, DSC, 2009.

[12] R GUI Projects, "Why a GUI?", *www.sciviews.org/_rgui/*, Philippe Grosjean, 2010.

[13] R Development Page, "Contributed R Packages provided by gWidgets" *www.r-forge.r-project.org/R/?group_id=761.*

[14] RwxWidgets, *www.omegahat.org/RwxWidgets/* , 2008.

[15] A quick start with RGtk2, which librarys you have to use and some demos,*www.ggobi.org/rgtk2/*